NHẬP MÔN JAVA

CHƯƠNG 4

HƯỚNG ĐỐI TƯỢNG

TRONG JAVA

MỞ ĐẦU

- Từ khi ra đời cho đến nay lập trình hướng đối tượng (OOP) đã chứng tỏ được sức mạnh, vai trò của nó trong các đề án tin học.
- Lập trình OOP là một phương pháp mạnh mẽ và rất hiệu quả để xây dựng nên những chương trình ứng dụng trên máy tính.
- Ở phần này chúng ta tìm hiểu các vấn đề cơ bản của lập trình hướng đối tượng trong Java thông qua việc tạo các lớp, các đối tượng và các tính chất của chúng.

PHẦN 1

LỚP

(CLASS)

KHÁI NIỆM LỚP (CLASS)

- Lớp được xem như một khuôn mẫu (template) của đối tượng (Object).
- Trong lớp bao gồm các thuộc tính của đối tượng (properties) và các phương thức (methods) tác động lên các thuộc tính.
- Đối tượng được xây dựng từ lớp nên được gọi là thể hiện của lớp (class instance).

KHAI BÁO LỚP

```
class <ClassName>
{
    <kiểu dữ liệu> <field_1>;
    <kiểu dữ liệu> <field_2>;
    constructor
    method_1
    method_2
}
```

- *class*: là từ khóa của java
- *ClassName*: là tên chúng ta đặt cho lớp
- *field_1, field_2*: các thuộc tính (các biến, hay các thành phần dữ liệu của lớp)
- *constructor*: là phương thức xây dựng, khởi tạo đối tượng của lớp.
- *method_1, method_2*: là các phương thức (có thể gọi là hàm) thể hiện các thao tác xử lý, tác động lên các thuộc tính của lớp.

THUỘC TÍNH CỦA LỚP

- Vùng dữ liệu (fields) hay thuộc tính (properties) của lớp được khai báo bên trong lớp như sau:

```
class <ClassName>
{
    // khai báo những thuộc tính của lớp
    <tiền tố> <kiểu dữ liệu> field1;
    // …
}
```

- Để xác định quyền truy xuất của các đối tượng khác đối với vùng dữ liệu của một lớp người ta thường dùng 3 tiền tố sau:
  - **public**: có thể truy xuất từ tất cả các đối tượng khác
  - **private**: một lớp không thể truy xuất vùng private của một lớp khác.
  - **protected**: vùng protected của một lớp chỉ cho phép bản thân lớp đó và những lớp dẫn xuất từ lớp đó truy cập đến.

THUỘC TÍNH CỦA LỚP

Ví dụ:

```
public class xemay
{    public String nhasx;
     public String model;
     private float chiphisx;
     protected int thoigiansx;
     // so luong so cua xe may: 3, 4
     protected int so;
     // sobanhxe là biến tĩnh có giá trị là 2 trong tất cả
     // các thể hiện tạo ra từ lớp xemay
     public static int sobanhxe = 2;
}
```

- Thuộc tính "*nhasx*", "*model*" có thể được truy cập đến từ tất cả các đối tượng khác.
- Thuộc tính "*chiphisx*" chỉ có thể truy cập được từ các đối tượng có kiểu "*xemay*"
- Thuộc tính "*thoigiansx*", so có thể truy cập được từ các đối tượng có kiểu "*xemay*" và các đối tượng của các lớp con dẫn xuất từ lớp "*xemay*"

THUỘC TÍNH CỦA LỚP

Lưu ý:

- Thông thường để an toàn cho vùng dữ liệu của các đối tượng người ta tránh dùng tiền tố public, mà thường chọn tiền tố private để ngăn cản quyền truy cập đến vùng dữ liệu của một lớp từ các phương thức bên ngoài lớp đó.

PHƯƠNG THỨC (METHOD) CỦA LỚP

- Hàm hay phương thức (method) trong Java là khối lệnh thực hiện các chức năng, các hành vi xử lý của lớp lên vùng dữ liệu.

Khai báo phương thức:

<Tiền tố> <kiểu trả về> <Tên phương thức> (<danh sách đối số>)

{

<khối lệnh>;

}

- Để xác định quyền truy xuất của các đối tượng khác đối với các phương thức của lớp người ta thường dùng các tiền tố sau:

public, protected, private, static, final, abstract, synchronized

- *<kiểu trả về>:* có thể là kiểu void, kiểu cơ sở hay một lớp.
- *<Tên phương thức>:* đặt theo qui ước giống tên biến.
- *<danh sách thông số>:* có thể rỗng

PHƯƠNG THỨC (METHOD) CỦA LỚP

- **public**: phương thức có thể truy cập được từ bên ngoài lớp khai báo.
- **protected**: có thể truy cập được từ lớp khai báo và những lớp dẫn xuất từ nó.
- **private**: chỉ được truy cập bên trong bản thân lớp khai báo.
- **static**: phương thức lớp dùng chung cho tất cả các thể hiện của lớp, có nghĩa là phương thức đó có thể được thực hiện kể cả khi không có đối tượng của lớp chứa phương thức đó.
- **final**: phương thức có tiền tố này không được khai báo chồng ở các lớp dẫn xuất.
- **abstract**: phương thức không cần cài đặt (không có phần source code), sẽ được hiện thực trong các lớp dẫn xuất từ lớp này.
- **synchoronized**: dùng để ngăn các tác động của các đối tượng khác lên đối tượng đang xét trong khi đang đồng bộ hóa. Dùng trong lập trình miltithreads.

THUỘC TÍNH CỦA LỚP

Ví dụ:

```
public class xemay
{   public String nhasx;
    public String model;
    private float chiphisx;
    protected int thoigiansx;
    // so luong so cua xe may: 3, 4 so
    protected int so;
    // là biến tĩnh có giá trị là 2 trong tất cả các thể hiện tạo ra từ lớp xemay
    public static int sobanhxe = 2;
    public float tinhgiaban() { return 1.5 * chiphisx; }
}
```

- **Lưu ý**:
  - Thông thường trong một lớp các phương thức nên được khai báo dùng từ khóa public, khác với vùng dữ liệu thường là dùng tiền tố private vì mục đích an toàn.
  - Những biến nằm trong một phương thức của lớp là các biến cục bộ (local) và nên được khởi tạo sau khi khai báo.

KHỞI TẠO MỘT ĐỐI TƯỢNG

- **Constructor** thật ra là một loại phương thức đặc biệt của lớp.
- Constructor dùng gọi tự động khi khởi tạo một thể hiện của lớp, có thể dùng để khởi gán những giá trị măc định.
- Các constructor không có giá trị trả về, và có thể có tham số hoặc không có tham số.
- Constructor phải có **cùng tên với lớp** và được gọi đến khi dùng từ khóa **new**.
- Nếu một lớp không có constructor thì Java sẽ cung cấp cho lớp một constructor mặc định (default constructor). Những thuộc tính, biến của lớp sẽ được khởi tạo bởi các giá trị mặc định (số: thường là giá trị 0, kiểu luận lý là giá trị false, kiểu đối tượng giá trị null, …)

Lưu ý: thông thường để an toàn, dễ kiểm soát và làm chủ mã nguồn chương trình chúng ta nên khai báo một constructor cho lớp.

VÍ DỤ VỀ CONSTRUCTOR

```
public class xemay
{   // …
    public xemay() { }
    public xemay(String s_nhasx, String s_model,
                 f_chiphisx, int i_thoigiansx, int i_so);
    {   nhasx = s_nhasx;
        model = s_model;
        chiphisx = f_chiphisx;
        thoigiansx = i_thoigiansx;
        so = i_so;
        // hoặc
        // this.nhasx = s_nhasx;
        // this.model = s_model;
        // this.chiphisx = f_chiphisx;
        // this.thoigiansx = i_thoigiansx;
        // this.so = i_so;
    }
}
```

BIẾN this

- Biến **this** là một biến ẩn tồn tại trong tất cả các lớp trong ngông ngữ Java. Một class trong Java luôn tồn tại một biến **this**.
- Biến **this** được sử dụng trong khi chạy và tham khảo đến bản thân lớp chứa nó.

Ví dụ:

```
<tiền tố> class A
{       <tiền tố> int <field_1>;
        <tiền tố> String <field_2>;
        // Contructor của lớp A
        public A(int par_1, String par_2)
        {
                this.field_1 = par_1;     this.field_2 = par_2;
        }

        <tiền tố> <kiểu trả về> <method_1>()
        { // …                }

        <tiền tố> <kiểu trả về> <method_2>()
        {
                this.method_1()
                // …
        }
}
```

KHAI BÁO CHỒNG PHƯƠNG THỨC

- Việc khai báo trong một lớp nhiều phương thức có cùng tên nhưng khác tham số (khác kiểu dữ liệu, khác số lượng tham số) gọi là khai báo chồng phương thức (overloading method).

Ví dụ:

```
public class xemay
{   // khai báo fields …
    public float tinhgiaban()
    {
        return 2 * chiphisx;
    }
    public float tinhgiaban(float huehong)
    {
        return (2 * chiphisx + huehong);
    }
}
```

PHẦN 2
ĐẶC ĐIỂM HƯỚNG ĐỐI TƯỢNG TRONG JAVA

ĐẶC ĐIỂM OOP TRONG JAVA

Để hỗ trợ những nguyên tắc cơ bản của lập trình hướng đối Tượng (OOP), tất cả các ngôn ngữ lập trình OOP, kể cả Java đều có ba đặc điểm chung:

- Tính đóng gói (Encapsulation).
- Tính đa hình (Polymorphimsm)
- Tính kế thừa (Inheritance)

TÍNH ĐÓNG GÓI (encapsulation)

- Cơ chế đóng gói trong lập trình hướng đối tượng giúp cho các đối tượng giấu đi một phần các chi tiết cài đặt, cũng như phần dữ liệu cục bộ của nó, và chỉ công bố ra ngoài những gì cần công bố để trao đổi với các đối tượng khác. Hay chúng ta có thể nói đối tượng là một thành tố hỗ trợ tính đóng gói.

- Đơn vị đóng gói cơ bản của ngôn ngữ java là class. Một class định nghĩa hình thức của một đối tượng. Một class định rõ những thành phần dữ liệu và các đoạn mã cài đặt các thao tác xử lý trên các đối tượng dữ liệu đó. Java dùng class để xây dựng những đối tượng. Những đối tượng là những thể hiện (instances) của một class.

- Một lớp bao gồm thành phần dữ liệu và thành phần xử lý.
  - Thành phần dữ liệu của một lớp thường bao gồm các biến thành viên và các biến thể hiện của lớp.
  - Thành phần xử lý là các thao tác trên các thành phần dữ liệu, thường trong Java người gọi là phương thức. Phương thức là một thuật ngữ hướng đối tượng trong Java. Ttrong C/C++ người ta dùng thuật ngữ là hàm hoặc phương thức.

TÍNH ĐA HÌNH (polymorphism)

- Hàm hay phương thức (method) trong Java là khối lệnh thực hiện các chức năng, các hành vi xử lý của lớp lên vùng dữ liệu.

```
class A_Object  {
        // …
        void method_1( ){  // … }
}
class B_Object extends A_Object {
        // …
        void method_1( ) {  // …}
}
class C {
        public static void main(String[] args)
        {     // Tạo một mảng 2 phần tử kiểu A
              A_Object arr_Object = new A_Object[2];
              B_Object var_1 = new B_Object();  arr_Object[0] = var_1;
              A_Object var_2;
              for (int i=0; i<2; i++) {          var_2 = arr_Object[i];
                                                 var_2.method_1();        }
        }
}
```

TÍNH ĐA HÌNH (polymorphysm)

Giải thích ví dụ của slide trước:

Vòng lặp for trong đoạn chương trình trên:

- Với i = 0 thì biến var_2 có kiểu là B_Object, và lệnh var_2.method_1() sẽ gọi thực hiện phương thức method_1 của lớp B_Object.
- Với i = 1 thì biến var_2 có kiểu là A_Object, và lệnh var_2.method_1() sẽ gọi thực hiện phương thức method_1 của lớp A_Object.

- Trong ví dụ trên đối tượng var_2 có thể nhận kiểu A_Object hay B_Object. Hay nói các khác, một biến đối tượng kiểu A_Object như var_2 trong ví dụ trên có thể tham chiếu đến bất kỳ đối tượng nào của bất kỳ lớp con nào của lớp A_Object
- Ví dụ, var_2 có thể tham chiếu đến đối tượng var_1, var_1 là đối tượng của lớp B_Object dẫn xuất từ lớp A_Object). Ngược lại một biến của lớp con không thể tham chiếu đến bất kỳ đối tượng nào của lớp cha.

TÍNH KẾ THỪA (inheritance)

- Một lớp con (subclass) có thể kế thừa tất cả những vùng dữ liệu và phương thức của một lớp khác (siêu lớp - superclass).
- Như vậy việc tạo một lớp mới từ một lớp đã biết sao cho các thành phần (fields và methods) của lớp cũ cũng sẽ thành các thành phần (fields và methods) của lớp mới. Khi đó ta gọi lớp mới là lớp dẫn xuất (derived class) từ lớp cũ (superclass).
- Có thể lớp cũ cũng là lớp được dẫn xuất từ một lớp nào đấy, nhưng đối với lớp mới vừa tạo thì lớp cũ đó là một lớp siêu lớp trực tiếp (immediate supperclass).

Dùng từ khóa **extends** để chỉ lớp dẫn xuất.

```
class A extends B
{
        // …
}
```

PHẦN 3
TÍNH KẾ THỪA

KHAI BÁO CHỒNG PHƯƠNG THỨC

- Tính kế thừa giúp cho các lớp con nhận được các thuộc tính/phương thức public và protected của lớp cha.
- Đồng thời cũng có thể thay thế các phương thức của lớp cha bằng cách khai báo chồng.

```
public class xega extends xemay
{
        public xega( ) { }
        public xega(String s_nhasx, String s_model, f_chiphisx, int i_thoigiansx);
        {    this.nhasx = s_nhasx;
             this.model = s_model;
             this.chiphisx = f_chiphisx;
             this.thoigiansx = i_thoigiansx;
             this.so = 0;
        }
        public float tinhgiaban( ) { return 2.5 * chiphisx; }
}
```

BA TIỀN TỐ TRONG KẾ THỪA

Java cung cấp 3 tiền tố để ***hỗ trợ tính kế thừa của lớp***:

- **public**: lớp có thể truy cập từ các gói, chương trình khác.
- **final**: Lớp hằng, lớp không thể tạo dẫn xuất (không thể có con), hay đôi khi người ta gọi là lớp "vô sinh".
- **abstract**: Lớp trừu tượng (không có khai báo các thành phần và các phương thức trong lớp trừu tượng). Lớp dẫn xuất sẽ khai báo, cài đặt cụ thể các thuộc tính, phương thức của lớp trừu tượng.

LỚP NỘI (INNER CLASS)

- Lớp nội là lớp được khai báo bên trong 1 lớp khác.
- Lớp nội thể hiện tính đóng gói cao và có thể truy xuất trực tiếp biến của lớp cha.

Ví dụ:

```
public class A {
        // …
        int <field_1>
        static class B {
                // …
                int <field_2>
                public B(int par_1) {
                        field_2 = par_1 + field_1;
                }
        }
}
```

- Trong ví dụ trên thì chương trình dịch sẽ tạo ra hai lớp với hai files khác nhau: A.class và B.class

LỚP VÔ SINH

- Lớp mà ta không thể có lớp dẫn xuất từ nó (không có lớp con) gọi là lớp "vô sinh", hay nói cách khác không thể kế thừa được từ một lớp "vô sinh". Lớp "vô sinh" dùng để hạn chế, ngăn ngừa các lớp khác dẫn xuất từ nó.
- Để khai báo một lớp là lớp "vô sinh", chúng ta dùng từ khóa **final class.**
- Tất cả các phương thức của lớp vô sinh đều vô sinh, nhưng các thuộc tính của lớp vô sinh thì có thể không vô sinh.

Ví dụ:

```
public final class A
{
        public final int x;
        private int y;
        public final void method_1()
        {
                // …
        }
        public final void method_2()
        {
                // …
        }
}
```

LỚP TRỪU TƯỢNG

- Lớp trừu tượng là lớp không có khai báo các thuộc tính thành phần và các phương thức.
- Các lớp dẫn xuất của nó sẽ khai báo thuộc tính, cài đặt cụ thể các phương thức của lớp trừu tượng.

Ví dụ:

```
abstract class A {
        abstract void method_1();
}

public class B extends A {
        public void method_1( ) {
                // cài đặt chi tiết cho phương thức method_1
                // trong lớp con B.
                // …
        }
}
```

```
public class C extends A
{
        public void method_1()
        {
                // cài đặt chi tiết cho
                //method_1 trong lớp con C.
                // …
        }
}
```

Lưu ý:

- Các phương thức được khai báo dùng các tiền tố **private** và **static** thì không được khai báo là trừu tượng **abstract**.
- Tiền tố **private** thì không thể truy xuất từ các lớp dẫn xuất, còn tiền tố **static** thì chỉ dùng riêng cho lớp khai báo mà thôi.

PHƯƠNG THỨC FINALIZE

- Trong java không có kiểu dữ liệu con trỏ như trong C, người lập trình không cần phải quá bận tâm về việc cấp phát và giải phóng vùng nhớ, sẽ có một trình dọn dẹp hệ thống đảm trách việc này. Trình dọn dẹp hệ thống sẽ dọn dẹp vùng nhớ cấp phát cho các đối tượng trước khi hủy một đối tượng.
- Phương thức *finalize()* là một phương thức đặc biệt được cài đặt sẵn cho các lớp. Trình dọn dẹp hệ thống sẽ gọi phương thức này trước khi hủy một đối tượng. Vì vậy việc cài đặt một số thao tác giải phóng, dọn dẹp vùng nhớ đã cấp phát cho các đối tượng dữ liệu trong phương thức *finalize()* sẽ giúp cho người lập trình chủ động kiểm soát tốt quá trình hủy đối tượng thay vị giao cho trình dọn dẹp hệ thống tự động. Đồng thời việc cài đặt trong phương thức *finalize()* sẽ giúp cho bộ nhớ được giải phóng tốt hơn, góp phần cải tiến tốc độ chương trình.

Ví dụ:

```
class A {
        // Khai báo các thuộc tính
        public void method_1( ) { // … }
        protected void finalize()
        {
                // Có thể dùng để đóng tất cả các kết nối
                // vào cơ sở dữ liệu trước khi hủy đối tượng.
                // …
        }
}
```

GÓI (package)

- Việc đóng gói các lớp lại tạo thành một thư viện dùng chung gọi là package.
- Một package có thể chứa một hay nhiều lớp bên trong, đồng thời cũng có thể chứa một package khác bên trong.
- Để khai báo một lớp thuộc một gói nào đấy ta phải dùng từ khóa **package**.
- Dòng khai báo gói phải là dòng đầu tiên trong tập tin khai báo lớp.
- Các tập tin khai báo lớp trong cùng một gói phải được lưu trong cùng một thư mục.
- **Lưu ý**: Việc khai báo import tất cả các lớp trong gói sẽ làm tốn bộ nhớ. Thông thường chúng ta chỉ nên import những lớp cần dùng trong chương trình.

Ví dụ:

```
package phuongtiengiaothong;
class xemay  { // …. }
class xega extends xemay { // …  }
```

- Khi đó muốn sử dụng lớp *xemay* vào chương trình ta sẽ khai báo như sau;

```
import phuongtiengiaothong.xemay;
```

GIAO DIỆN (interface)

Khái niệm interface:

- Như chúng ta đã biết một lớp trong java chỉ có một siêu lớp trực tiếp hay một cha duy nhất (đơn thừa kế).
- Để tránh đi tính phức tạp của đa thừa kế (multi-inheritance) trong lập trình hướng đối tượng, Java thay thế bằng giao tiếp (interface).
- Một lớp có thể có nhiều giao tiếp (interface) với các lớp khác để thừa hưởng thêm vùng dữ liệu và phương thức của các giao tiếp này.

Khai báo interface:

- Interface được khai báo như một lớp. Nhưng các thuộc tính của interface là các hằng (khai báo dùng từ khóa final) và các phương thức của giao tiếp là trừu tượng (mặc dù không có từ khóa abstract).
- Trong các lớp có cài đặt các interface ta phải tiến hành cài đặt cụ thể các phương thức này.

GIAO DIỆN (interface)

Ví dụ:

```
public interface sanpham
{    static final String nhasx = “Honda VN”;
     static final String dienthoai = “08-8123456”;
     public int gia(String s_model);
}

// khai báo 1 lớp có cài đặt interface
public class xemay implements sanpham
{    // cài đặt lại phương thức của giao diện trong lớp
     public int gia(String s_model)
     {
          if (s_model.equals(“2005”))  return (2000);
          else  return (1500);
     }
     public String chobietnhasx() {  return (nhasx); }
}
```

GIAO DIỆN (interface)

- Có một vấn đề khác với lớp là một giao diện (interface) không chỉ có một giao diện cha trực tiếp mà có thể dẫn xuất cùng lúc nhiều giao diện khác (hay có nhiều giao diện cha).
- Khi đó nó sẽ kế thừa tất cả các giá trị hằng và các phương thức của các giao diện cha.
- Các giao diện cha được liệt kê thành chuỗi và cách nhau bởi dấu phẩy “,”.

Khai báo như sau:

```
public interface InterfaceName extends interface1, interface2, interface3
{
        // …
}
```

PHẦN 4
MỘT SỐ VÍ DỤ

VÍ DỤ 1: VỀ TÍNH ĐA HÌNH (lớp Shape)

- Sau đây là minh họa tính đa hình (polymorphism) trong phân cấp kế thừa thông qua việc mô tả và xử lý một số thao tác cơ bản trên các đối tượng hình học.

```
// Định nghĩa lớp trừu tượng cơ sở tên Shape trong tập tin Shape.java
public abstract class Shape extends Object
{
    // trả về diện tích của một đối tượng hình học shape
    public double area( ) { return 0.0; }
    // trả về thể tích của một đối tượng hình học shape
    public double volume( ) { return 0.0; }
    // Phương thức trừu tượng cần phải được hiện thực
    // trong những lớp con để trả về tên đối tượng
    // hình học shape thích hợp
    public abstract String getName();
} // end class Shape
```

VÍ DỤ 1: VỀ TÍNH ĐA HÌNH (lớp Point)

Định nghĩa lớp Point trong tập tin Point.java. Lớp Point thừa kế lớp Shape

```
public class Point extends Shape
{    protected int x, y; // Tọa độ x, y của 1 điểm
     public Point( ){ setPoint( 0, 0 ); } // constructor không tham số.
     public Point(int xCoordinate, int yCoordinate) // constructor có tham số.
     { setPoint( xCoordinate, yCoordinate ); }
     public void setPoint( int xCoordinate, int yCoordinate )// gán tọa độ x, y cho
     1 điểm
     { x = xCoordinate; y = yCoordinate; }
     public int getX( ) {  return x; } // lấy tọa độ x của 1 điểm
     public int getY( ) { return y; } // lấy tọa độ y của 1 điểm
     public String toString() // Thể hiện tọa độ của 1 điểm dưới dạng chuỗi
     {  return "[" + x + ", " + y + "]"; }
     public String getName() // trả về tên của đối tượng shape
     {  return "Point"; }
} // end class Point
```

GIẢI THÍCH (Shape và Point)

- Định nghĩa một lớp cha **Shape** là một lớp trừu tượng dẫn xuất từ Object và có 3 phương thức khai báo dùng tiền tố public.
- Phương thức **getName()** khai báo trừu tượng vì vậy nó phải được hiện thực trong các lớp con. Phương thức **area()** (tính diện tích) và phương thức **volume()** (tính thể tích) được định nghĩa và trả về 0.0.
- Những phương thức này sẽ được khai báo chồng trong các lớp con để thực hiện chức năng tính diện tích cũng như thể tích phù hợp với những đối tượng hình học tương ứng (đường tròn, hình trụ, …)
- Lớp **Point:** dẫn xuất từ lớp **Shape**. Một điểm thì có diện tích và thể tích là 0.0, vì vậy những phương thức **area**() và **volume**() của lớp cha không cần khai báo chồng trong lớp **Point,** chúng được thừa kế như đã định nghĩa trong lớp trừu tượng **Shape**.
- Những phương thức khác như **setPoint**(…) để gán tọa độ x, y cho một điểm, còn phương thức **getX()**, **getY()** trả về tọa độ x, y của một điểm. Phương thức **getName()** là hiện thực cho phương thức trừu tượng trong lớp cha, nếu như phương thức **getName()** mà không được định nghĩa thì lớp **Point** là một lớp trừu tượng.

VÍ DỤ 1: VỀ TÍNH ĐA HÌNH (lớp Circle)

Định nghĩa lớp Circle trong tập tin Circle.java. Lớp Circle thừa kế lớp Point

```
public class Circle extends Point      // Dẫn xuất từ lớpPoint
{       protected double radius;
        public Circle()  // constructor không tham số
        {
              // ngầm gọi đến constructor của lớp cha
              setRadius( 0 );
        }
        public Circle( double circleRadius, int xCoordinate, int yCoordinate )
        {      // constructor có tham số
              // gọi constructorcủa lớp cha
              super( xCoordinate, yCoordinate );
              setRadius( circleRadius );
        }
        public void setRadius( double circleRadius ) // Gán bán kính của đường tròn
        {
              radius = ( circleRadius >= 0 ? circleRadius:0 );
        }
        //còn tiếp ở slide sau
```

VD1: VỀ TÍNH ĐA HÌNH (lớp Circle - tt)

```
    public double getRadius() // Lấy bán kính của đường tròn
    {
        return radius;
    }
    public double area()        // Tính diện tích đường tròn Circle
    {
        return Math.PI * radius * radius;
    }
    public String toString()    // Biểu diễn đường tròn bằng một chuỗi
    {
        return "Center = " + super.toString()  + "; Radius = " + radius;
    }
    public String getName() // trả về tên của shape
    {
        return "Circle";
    }
} // end class Circle
```

GIẢI THÍCH (lớp Circle)

- Lớp **Circle** dẫn xuất từ lớp **Point**, một đường tròn có thể tích là 0.0, vì vậy phương thức **volume()** của lớp cha không khai báo chồng, nó sẽ thừa kế từ lớp **Point**, mà lớp **Point** thì thừa kế từ lớp **Shape**.
- Diện tích đường tròn khác với một điểm, vì vậy phương thức tính diện tích **area()** được khai báo chồng.
- Phương thức **getName()** hiện thực phương thức trừu tượng đã khai báo trong lớp cha, nếu phương thức **getName()** không khai báo trong lớp **Circle** thì nó sẽ kế thừa từ lớp **Point**.
- Phương thức **setRadius** dùng để gán một bán kính (radius) mới cho một đối tượng đường tròn, còn phương thức **getRadius** trả về bán kính của một đối tượng đường tròn.

VÍ DỤ 1: VỀ TÍNH ĐA HÌNH (lớp Cylinder)

Định nghĩa lớp Cylinder trong tập tin Cylinder.java. Lớp Cylinder thừa kế từ lớp Circle

```
public class Cylinder extends Circle
{       protected double height; // chiều cao của Cylinder
        public Cylinder() // constructor không có tham số
        {
            // ngầm gọi đến constructor của lớp cha
            setHeight( 0 );
        }
        // constructor có tham số
        public Cylinder( double cylinderHeight, double cylinderRadius,
                         int xCoordinate,int yCoordinate )
        {
            // Gọi constructor của lớp cha
            super( cylinderRadius, xCoordinate, yCoordinate );
            setHeight( cylinderHeight );
        }
```

//còn tiếp ở slide sau

VD1: VỀ TÍNH ĐA HÌNH (lớp Cylinder-tt)

```
public void setHeight( double cylinderHeight )
{     // Gán chiều cao cho Cylinder
      height = ( cylinderHeight >= 0 ? cylinderHeight:0 );
}
public double getHeight() // Lấy chiều cao của Cylinder
{     return height; }
public double area() // Tính diện tích xung quanh của Cylinder
{     return 2 * super.area() + 2 * Math.PI * radius *height; }
public double volume() // Tính thể tích của Cylinder
{ return super.area()  * height;}
public String toString() // Biểu diễn Cylinder bằng một chuỗi
{
      return super.toString()  + "; Height = " + height;
}

public String getName() // trả về tên của shape
{
      return "Cylinder";
}
} // end class Cylinder
```

GIẢI THÍCH (lớp Cylinder)

- Lớp **Cylinder** dẫn xuất từ lớp Circle. Một Cylinder (hình trụ) có diện tích và thể tích khác với một Circle (hình tròn), vì vậy cả hai phương thức **area()** và **volume()** cần phải khai báo chồng.
- Phương thức getName() là hiện thực phương thức trừu tượng trong lớp cha, nếu phương thức **getName()** không khai báo trong lớp **Cylinder** thì nó sẽ kế thừa từ lớp **Circle**.
- Phương thức **setHeight** dùng để gán chiều cao mới cho một đối tượng hình trụ.
- Còn phương thức **getHeight** trả về chiều cao của một đối tượng hình trụ.

VÍ DỤ 1: VỀ TÍNH ĐA HÌNH (test)

File Test.java dùng để kiểm tra tính kế thừa của Point, Circle, Cylinder với lớp trừu tượng Shape.

```
import java.text.DecimalFormat;
public class Test
{      // Kiểm tra tính kế thừa của các đối tượng hình học
       public static void main( String args[] )
       {    // Tạo ra các đối tượng hìnhhọc
            Point point = new Point( 7, 11 );
            Circle circle = new Circle( 3.5, 22, 8 );
            Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );
            // Tạo một mảng các đối tượng hình học
            Shape arrayOfShapes[] = new Shape[ 3 ];
            // arrayOfShapes[ 0 ] là một đối tượng Point
            arrayOfShapes[ 0 ] = point;
            // arrayOfShapes[ 1 ] là một đối tượng Circle
            arrayOfShapes[ 1 ] = circle;
            // arrayOfShapes[ 2 ] là một đối tượng cylinder
            arrayOfShapes[ 2 ] = cylinder;
```

//còn tiếp ở slide sau

VÍ DỤ 1: VỀ TÍNH ĐA HÌNH (test - tt)

```
    // Lấy tên và biểu diễn của mỗi đối tượng hình học
    String output = point.getName() + ": " + point.toString() + "\n" +
        circle.getName() + ": " + circle.toString() + "\n" +
        cylinder.getName() + ": " + cylinder.toString();
    DecimalFormat precision2 = new DecimalFormat("0.00" );
    // duyệt mảng arrayOfShapes lấy tên, diện tích, thể tích
    // của mỗi đối tượng hình học trong mảng.
    for ( int i = 0; i < arrayOfShapes.length; i++ )
    {
        output += "\n\n" + arrayOfShapes[ i ].getName() +": "
                  + arrayOfShapes[ i].toString() +"\n Area = "
                  + precision2.format( arrayOfShapes[ i ].area() ) +"\nVolume = "
                  + precision2.format( arrayOfShapes[ i ].volume() );
    }
    System.out.println(output);
    System.exit( 0 );
  }
}// end class Test
```

VD2: VỀ TÍNH ĐA HÌNH (interface Shape)

Tương tự ví dụ 1 nhưng trong ví dụ 2 chúng ta dùng interface để định nghĩa cho **Shape** thay vì một lớp trừu tượng. Vì vậy tất cả các phương thức tron interface **Shape** phải được hiện thực trong lớp **Point** là lớp cài đặt trực tiếp interface **Shape**.

```
// Định nghĩa một interface Shape trong tập tin shape.java
public interface Shape
{
        // Tính diện tích
        public abstract double area();
        // Tính thể tích
        public abstract double volume();
        // trả về tên của shape
        public abstract String getName();
}
```

VD2: VỀ TÍNH ĐA HÌNH (lớp Point)

Lớp *Point* cài đặt, hiện thực *interface* tên *shape*. *Định nghĩa lớp Point trong tập tin Point.java*

```
public class Point extends Object implements Shape
{
    protected int x, y; // Tọa độ x, y của 1 điểm
    public Point() // constructor không tham số.
    {
        setPoint( 0, 0 );
    }
    public Point(int xCoordinate, int yCoordinate) // constructor có tham số.
    {
        setPoint( xCoordinate, yCoordinate );
    }
    public void setPoint( int xCoordinate, int yCoordinate ) // gán tọa độ x, y cho 1 điểm
    {   x = xCoordinate;
        y = yCoordinate;
    }
//còn tiếp ở slide sau
```

VD2: VỀ TÍNH ĐA HÌNH (lớp Point - tt)

```
        // lấy tọa độ x của 1 điểm
        public int getX() {  return x; }
        public int getY() // lấy tọa độ y của 1 điểm
        { return y; }
        public String toString() // Thể hiện tọa độ của 1 điểm dưới dạng chuỗi
        { return "[" + x + ", " + y + "]"; }
        public double area() // Tính diện tích
        {  return 0.0; }
        public double volume() // Tính thể tích
        { return 0.0; }
        public String getName() // trả về tên của đối tượng shape
        {
              return "Point";
        }
} // end class Point
```

VD2: VỀ TÍNH ĐA HÌNH (lớp Circle)

Lớp *Circle* là lớp con của lớp *Point*, và cài đặt/hiện thực gián tiếp *interface* tên *shape*. *Định nghĩa lớp Circle trong tập tin Circle.java*

```
public class Circle extends Point
{       // Dẫn xuất từ lớpPoint
        protected double radius;
        public Circle() // constructor không tham số
        {
              // ngầm gọi đến constructor của lớp cha
        setRadius( 0 );
        }
        // constructor có tham số
        public Circle( double circleRadius, int xCoordinate, int yCoordinate )
        {
              // gọi constructorcủa lớp cha
              super( xCoordinate, yCoordinate );
              setRadius( circleRadius );
        }
```

//còn tiếp ở slide sau

VD2: VỀ TÍNH ĐA HÌNH (lớp Circle - tt)

```
    public void setRadius( double circleRadius ) // Gán bán kính của đường tròn
    {    radius = ( circleRadius >= 0 ? circleRadius:0 ); }
    public double getRadius() // Lấy bán kính của đường tròn
    { return radius; }
    public double area() // Tính diện tích đường tròn Circle
    { return Math.PI * radius * radius; }
    public String toString() // Biểu diễn đường tròn bằng một chuỗi
    {
        return "Center = " + super.toString() + "; Radius = " + radius;
    }
    // trả về tên của shape
    public String getName()
    {
        return "Circle";
    }
} // end class Circle
```

VD2: VỀ TÍNH ĐA HÌNH (lớp Cylinder)

Định nghĩa lớp hình trụ Cylinder trong tập tin Cylinder.java.

```
public class Cylinder extends Circle
{       protected double height; // chiều cao của Cylinder
        public Cylinder() // constructor không có tham số
        {
              // ngầm gọi đến constructor của lớp cha
              setHeight( 0 );
        }
        // constructor có tham số
        public Cylinder( double cylinderHeight, double cylinderRadius,
                         int xCoordinate,int yCoordinate )
        {     // Gọi constructor của lớp cha
              super( cylinderRadius, xCoordinate,yCoordinate );
              setHeight( cylinderHeight );
        }
        public void setHeight( double cylinderHeight ) // Gán chiều cao cho Cylinder
        { height = ( cylinderHeight >= 0 ? cylinderHeight:0 ); }
//còn tiếp ở slide sau
```

VD2: VỀ TÍNH ĐA HÌNH (lớp Cylinder-tt)

```
    public double getHeight() // Lấy chiều cao của Cylinder
    {  return height; }
    public double area() // Tính diện tích xung quanh của Cylinder
    { return 2 * super.area() + 2 * Math.PI * radius * height; }
    public double volume() // Tính thể tích của Cylinder
    { return super.area() * height; }
    public String toString() // Biểu diễn Cylinder bằng một chuỗi
    {
        return super.toString() + "; Height = " + height;
    }
    // trả về tên của shape
    public String getName()
    {
        return "Cylinder";
    }
} // end class Cylinder
```

VD2: VỀ TÍNH ĐA HÌNH (test)

Tập tin Test.java để kiểm tra tính kế thừa của Point, Circle, Cylinder với interface Shape.

```
import java.text.DecimalFormat;
public class Test // Kiểm tra tính kế thừa của các đối tượng hình học
{       public static void main( String args[] )
        {     // Tạo ra các đối tượng hìnhhọc
              Point point = new Point( 7, 11 );
              Circle circle = new Circle( 3.5, 22, 8 );
              Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );
              Shape arrayOfShapes[] = new Shape[ 3 ]; // Tạo một mảng các đối tượng hình học
              // arrayOfShapes[ 0 ] là một đối tượng Point
              arrayOfShapes[ 0 ] = point;
              arrayOfShapes[ 1 ] = circle; // arrayOfShapes[ 1 ] là một đối tượng Circle
              arrayOfShapes[ 2 ] = cylinder; // arrayOfShapes[ 2 ] là một đối tượng cylinder
              // Lấy tên và biểu diễn của mỗi đối tượng hình học
              String output = point.getName() + ": " + point.toString() + "\n"
                            + circle.getName() + ": " + circle.toString() + "\n“
                            + cylinder.getName() + ": " + cylinder.toString();
              DecimalFormat precision2 = new DecimalFormat("0.00" );
//còn tiếp ở slide sau
```

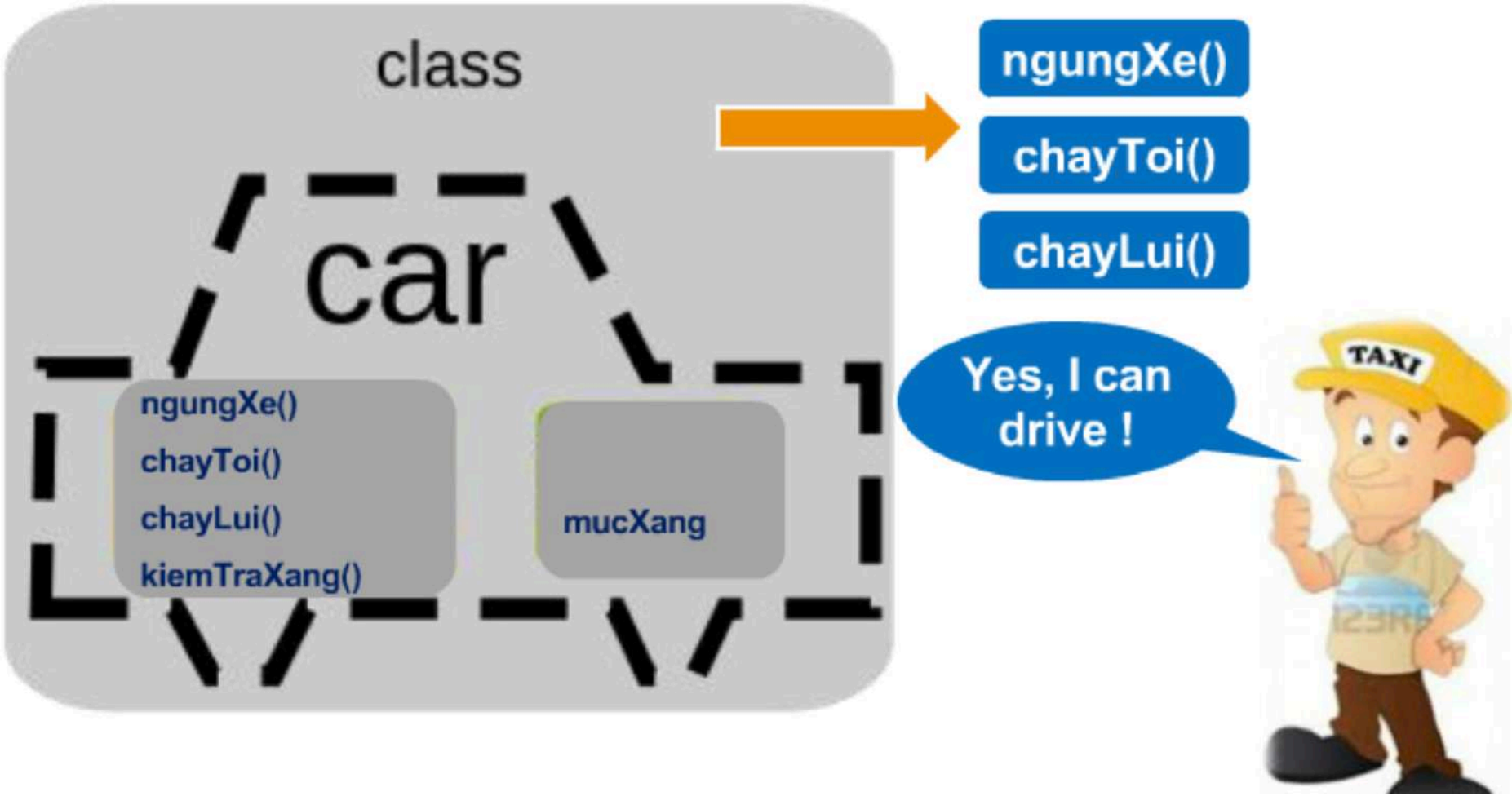
VD2: VỀ TÍNH ĐA HÌNH (test - tt)

```
        // duyệt mảng arrayOfShapes lấy tên, diện tích, thể tích
        // của mỗi đối tượng hình học trong mảng.
        for ( int i = 0; i < arrayOfShapes.length; i++ )
        {
            output += "\n\n" + arrayOfShapes[ i ].getName()
                      + ": " + arrayOfShapes[ i].toString() + "\n Area = "
                      + precision2.format( arrayOfShapes[ i ].area() )
                      + "\nVolume = "
                      + precision2.format( arrayOfShapes[ i ].volume() );
        }
        System.out.println(output);
        System.exit( 0 );
    }
} // end class Test
```

Tổng kết

Tóm lược

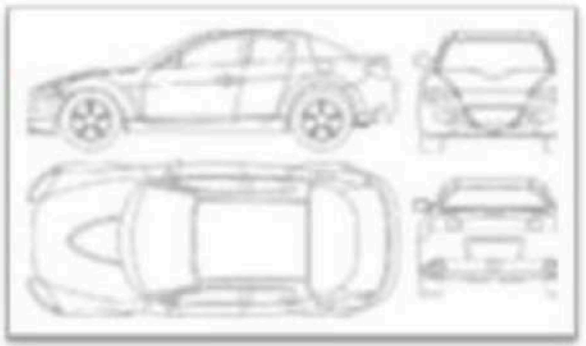
- Tính đóng gói (Encapsulation).



Tóm lược

- Tính đa hình (Polymorphimsm)

Trừu tượng hóa



Tóm lược

- Tính kế thừa (Inheritance)





Tóm lược

Truy suất

Chỉ định từ truy xuất	Các thành viên sẽ thấy trong			
	Class	Package	Lớp con (trong package khác)	Ngoài
public	yes	yes	yes	yes
protected	yes	yes	yes	no
private	yes	no	no	no
Không có chỉ định từ truy xuất	yes	yes	no	no

HẾT
CHƯƠNG 4