Trường Cao Đẳng CNTT TP.HCM
Khoa Công Nghệ Thông Tin

egenera

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

egenera

HƯỚNG ĐỐI TƯỢNG

1. LỚP
2. ĐẶC ĐIỂM HƯỚNG ĐỐI TƯỢNG TRONG JAVA
3. TÍNH KẾ THỪA
4. ĐA HÌNH VÀ TRỪU TƯỢNG
5. GIAO DIỆN

LỚP

1. KHÁI NIỆM

✓ **Lớp đối tượng (class):**

Định nghĩa các thuộc tính (đặc điểm) và hành động (phương thức) chung cho tất cả các đối tượng của cùng một loại.

✓ **Đối tượng (object):**

Thể hiện cụ thể của một lớp đối tượng.

✓ **Ví dụ:**

- Lớp (**class**) SINHVIEN có

  - **Thuộc tính:** Họ tên, giới tính, ngày tháng năm sinh, điểm trung bình, đối tượng ưu tiên, ...

  - **Phương thức:** Học bài, làm bài thi, làm bài tập, ...

- Sinh viên Nguyễn Văn A, Lý Thị B là đối tượng (**object**) thuộc lớp SINHVIEN

1. ĐỊNH NGHĨA ĐỐI TƯỢNG

Cú pháp:

```
class <Tên lớp>
{
        <từ khóa truy xuất> thuộc tính;
        <từ khóa truy xuất> phương thức();
        {
                Cài đặt;
        }
}
```

1. ĐỊNH NGHĨA ĐỐI TƯỢNG

Ví dụ: Định nghĩa lớp HocSinh

```
class HOCSINH
{
    private string hoten;
    private int toan, van;
    private float dtb;
    public void  Nhap()
    {
        // Cài đặt
    }
    public void  Xuat()
    {
        //Cài đặt
    }
}
```

1. TẠO VÀ SỬ DỤNG ĐỐI TƯỢNG

1. Tạo đối tượng

Cú pháp:

<Tên lớp> tên đối tượng = new <Tên lớp>();

Ví dụ:

```
HOCSINH hsA = new HOCSINH();
```

2. Sử dụng đối tượng

Cú pháp:

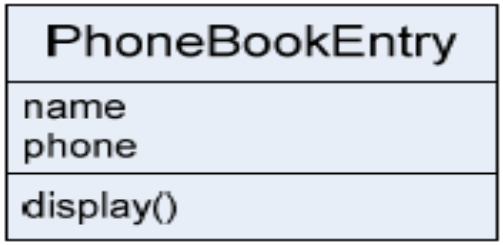
```
Tên_đối_tượng.Tên_phương_thức([tham số]);
```

Ví dụ:

```
hsA.Nhap();
hsA.Xuat();
```

1. TẠO VÀ SỬ DỤNG ĐỐI TƯỢNG

Ví dụ: Nhập vào họ tên, số điện thoại, hiển thị thông tin vừa nhập




```
class PhoneBookEntry {
  String name;
  String phone;

  void display() {
    System.out.println("Name: " + name);
    System.out.println("Phone: " + phone);
  }
}
```

```
public class PhoneBookTestDrive {
  public static void main (String[] args) {
    PhoneBookEntry tom = new PhoneBookEntry();
    tom.name = "Tom the Cat";
    tom.phone = "84208594";
    PhoneBookEntry jerry = new PhoneBookEntry();
    jerry.name = "Jerry the Mouse";
    jerry.phone = "98768065";

    tom.display();
    jerry.display();
  }
}
```

1. KHÁI NIỆM LỚP (CLASS)

- Lớp được xem như một khuôn mẫu (template) của đối tượng (Object).
- Trong lớp bao gồm các thuộc tính (properties) của đối tượng 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).

1. KHÁI NIỆM LỚP (CLASS)

Lớp (Class)

XE MÁY
Màu sắc Loại Model Thời gian sản xuất Chi phí sản xuất Số bánh xe
Tính giá bán()

Đối tượng (instance)

Piaggio
Trắng Tay ga Piaggio Fly 150 2012 2.000 2
Tính giá bán()

Lead
Đỏ Tay ga Honda Lead 2011 2011 1.700 2
Tính giá bán()

1. 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.

1. 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.

1. THUỘC TÍNH CỦA LỚP

Ví dụ:

```
public class xemay
{    public String mausac;
     public String loai
     public String model;
     private float chiphisx;
     protected int thoigiansx;
     public int sobanhxe;
}
```

- Thuộc tính "*mausac*", "*loai*", "*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*"

1. 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.

1. 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

1. 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.

1. PHƯƠNG THỨC (METHOD) CỦA LỚP

Ví dụ:

```
public class xemay
{    public String mausac;
     public String Loai
     public String model;
     private float chiphisx;
     protected int thoigiansx;
     public int sobanhxe;
     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.

1. 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.

1. VÍ DỤ VỀ CONSTRUCTOR

```
public class xemay
{   // …
    public xemay() { }
    public xemay(String s_mausac, String s_model,
                 int i_thoigiansx, f_chiphisx, int i_sobanhxe);
    {   mausac = s_mausac;
        model = s_model;
        chiphisx = f_chiphisx;
        thoigiansx = i_thoigiansx;
        sobanhxe = i_sobanhxe;
        // hoặc
        //this.mausac = s_mausac;
        //this.model = s_model;
        //this.chiphisx = f_chiphisx;
        //this.thoigiansx = i_thoigiansx;
        //this.sobanhxe = i_sobanhxe;
        }
}
```

1. VÍ DỤ VỀ CONSTRUCTOR

xemay Piaggio = new xemay();

Piaggio
null null null 0 0 0
Tính giá bán()

xemay Lead = new xemay(“đỏ”, ”Honda Lead 2011”,2011,1700,2);

Lead
Đỏ null Honda Lead 2011 2011 1.700 2
Tính giá bán()

1. 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()
              // …
        }
}
```

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);
     }
}
```

1. TÍNH CHẤT TĨNH

Có những trường hợp mà ta muốn tất cả các đối tượng thuộc một lớp chia sẻ một giá trị của một biến cụ thể thay vì phải từng đối tượng duy trì bản sao của chính mình của biến đó như là một thuộc tính. Ngôn ngữ Java đáp ứng nhu cầu này thông qua các thuộc tính tĩnh **(static)** có liên quan đến tất cả đối tượng của lớp đó hơn là với các cá nhân đối tượng riêng lẽ.

Khi bạn khai báo một biến là **static**, thì biến đó được gọi là biến tĩnh, hay biến **static**.

Biến static có thể được sử dụng để tham chiếu thuộc tính chung của tất cả đối tượng (mà không là duy nhất cho mỗi đối tượng), ví dụ như tên công ty của nhân viên, tên trường học của các sinh viên, …

1. TÍNH CHẤT TĨNH

Ví dụ: Biến static trong java

```
//Chuong trinh vi du ve bien static trong Java

class Student8{
   int rollno;
   String name;
   static String college ="BachKhoa";

   Student8(int r,String n){
   rollno = r;
   name = n;
   }
 void display (){System.out.println(rollno+" "+name+" "+college);}

 public static void main(String args[]){
 Student8 s1 = new Student8(111,"Hoang");
 Student8 s2 = new Student8(222,"Thanh");

 s1.display();
 s2.display();
 }
}
```

1. TÍNH CHẤT TĨNH

Ví dụ: CT không sử dụng biến static

```
class Counter{
int count=0; //se lay bo nho (memory) khi bien instance duoc tao
 //Ket qua thuc hien chuong trinh hien ra 3 so 1 o 3 dong
Counter(){
count++;
System.out.println(count);
}

public static void main(String args[]){

Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();

 }
}
```

Ví dụ: CT Sử dụng biến static

```
class Counter2{
static int count=0; //se lay bo nho chi mot lan và giu lai gia tri cua no
 //ket qua thuc hien in ra 3 dong cac gia tri : 1,2,3
Counter2(){
count++;
System.out.println(count);
}

public static void main(String args[]){

Counter2 c1=new Counter2();
Counter2 c2=new Counter2();
Counter2 c3=new Counter2();

 }
}
```

1. Phương thức tĩnh

Nếu bạn áp dụng từ khóa static với bất cứ phương thức nào, thì phương thức đó được gọi là phương thức static.

- Một phương thức static thuộc lớp chứ không phải đối tượng của lớp.
- Một phương thức static có thể được triệu hồi mà không cần tạo một instance của một lớp.
- Phương thức static có thể truy cập thành viên dữ liệu static và có thể thay đổi giá trị của nó.

1. Phương thức tĩnh

Ví dụ: Phương thức tĩnh

```
class Student9{
     int rollno;
     String name;
     static String college = "BachKhoa";

     static void change(){
     college = "QuocGia";
     }

     Student9(int r, String n){
     rollno = r;
     name = n;
     }

     void display (){System.out.println(rollno+" "+name+" "+college);}

    public static void main(String args[]){
    Student9.change();

    Student9 s1 = new Student9 (111,"Hoang");
    Student9 s2 = new Student9 (222,"Thanh");
    Student9 s3 = new Student9 (333,"Nam");

    s1.display();
    s2.display();
    s3.display();
    }
}
```

ĐẶC ĐIỂM HƯỚNG ĐỐI TƯỢNG TRONG JAVA

2. 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)

2. 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 KẾ THỪA

2. 1TÍNH KẾ THỪA (inheritance)

Khảo sát các lớp sau:

Sinh viên
Tên Địa chỉ Điểm môn 1 Điểm môn 2
Nhập tên () Nhập địa chỉ() Nhập điểm() Tính tổng điểm()

Nhân viên
Tên Địa chỉ Lương Chức vụ
Nhập tên () Nhập địa chỉ() Nhập chức vụ() Tính lương()

Khách hàng
Tên Địa chỉ Kiểu xe đã bán
Nhập tên () Nhập địa chỉ() Nhập kiểu xe() Xuất hóa đơn()

Trong cả ba lớp trên, chúng ta thấy có một vài thuộc tính và hoạt động chung. Chúng ta muốn nhóm những thuộc tính và những hoạt động ấy lại và định nghĩa chúng trong một lớp Người.

Người
Tên Địa chỉ
Nhập tên () Nhập địa chỉ()

2.1 TÍNH KẾ THỪA (inheritance)

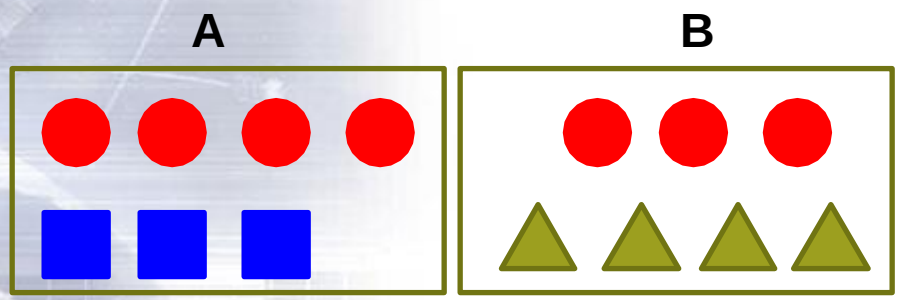
Khi đó ta thấy ba lớp “Sinh viên”, “Nhân viên”, “Khách hàng” đều có tất cả các thuộc tính và phương thức của lớp “Người” và có những thuộc tính và phương thức riêng của mỗi lớp.

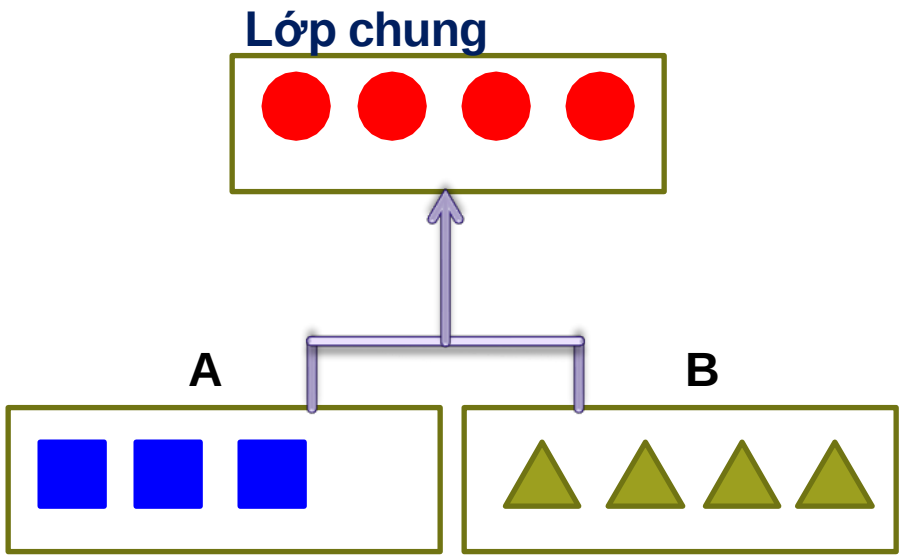
Như vậy, chúng ta cần khai báo lớp “Người” và sử dụng nó trong khi định nghĩa các lớp “Sinh viên”, “Nhân viên”, “Khách hàng”. Ta nói lớp “Sinh viên”, “Nhân viên”, “Khách hành” kế thừa lớp “Người”

Tính kế thừa cho phép một lớp chia sẻ các thuộc tính và phương thức được định nghĩa trong một hoặc nhiều lớp khác

2.1 KHÁI NIỆM

- Kế thừa thể hiện khả năng tái sử dụng các lớp đã được định nghĩa.
- Có thể định nghĩa lớp đối tượng mới dựa trên 1 hay nhiều lớp đối tượng đã có sẵn.
- Lớp có sẵn được gọi là lớp cơ sở (based class) và lớp kế thừa được gọi là lớp dẫn xuất (derived class)





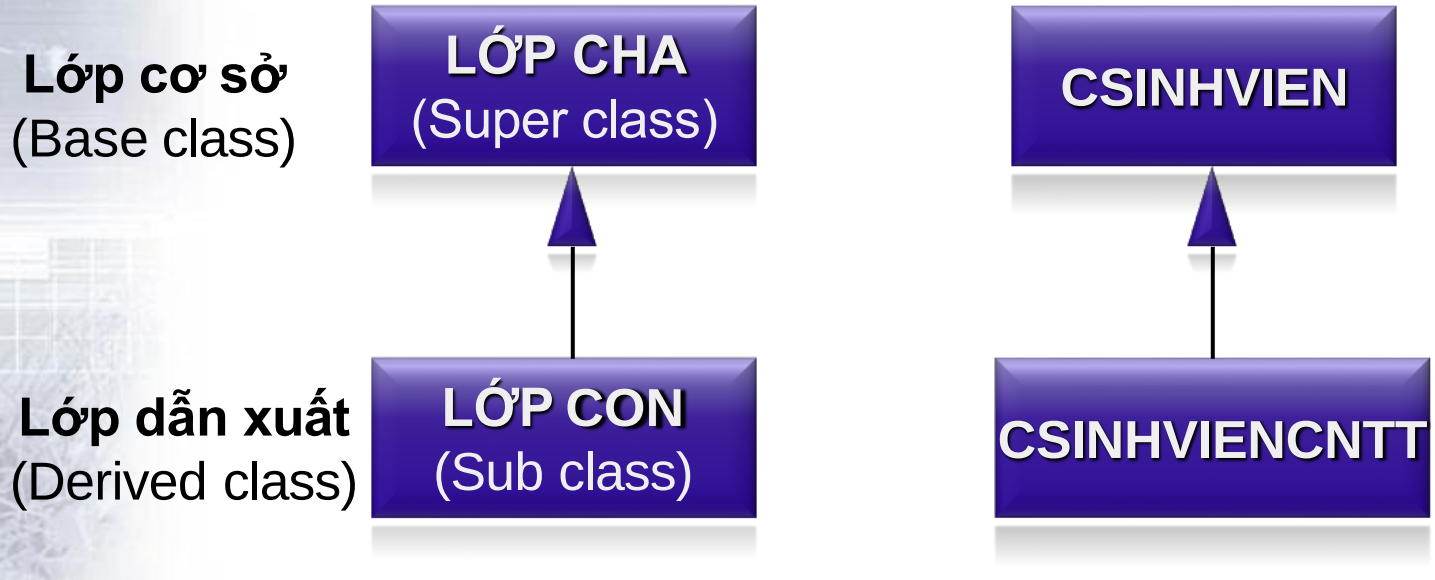
Thuộc tính chung

Thuộc tính riêng của A

Thuộc tính riêng của B

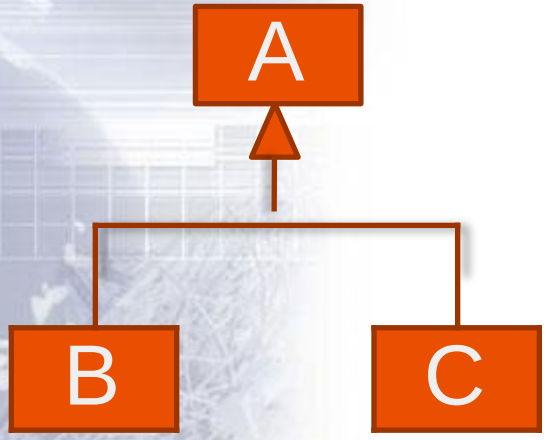
2.1. KHÁI NIỆM

- Kế thừa từ các lớp có từ trước.
- Ích lợi: có thể tận dụng lại
  - Các thuộc tính chung
  - Các hàm có thao tác tương tự



2.1 CÁC KÝ HIỆU

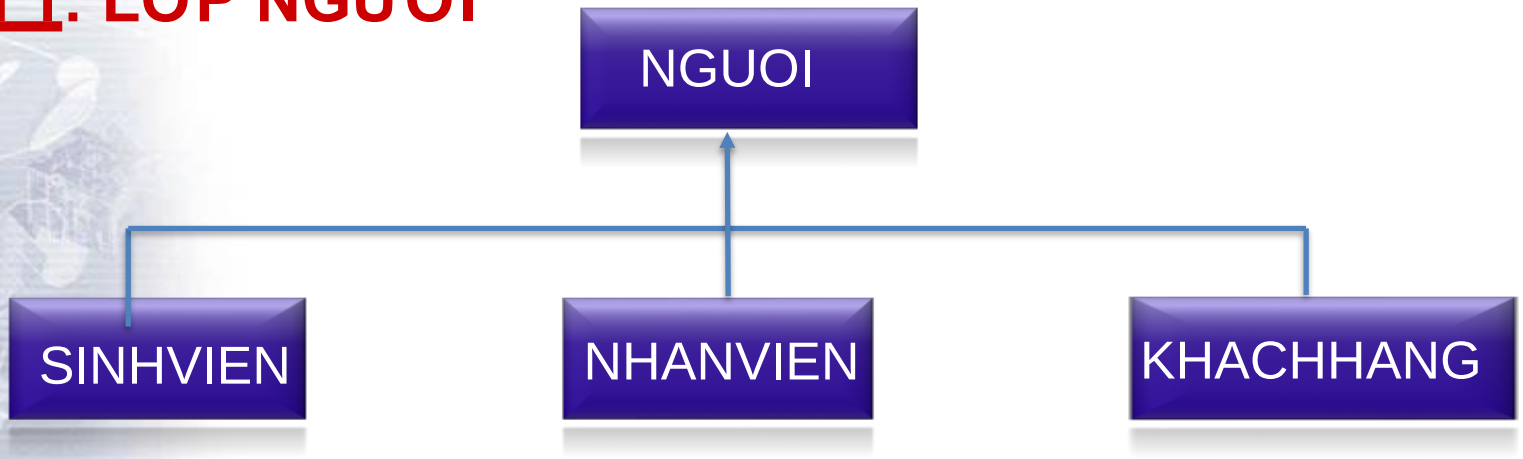
- ✓Là trường hợp tổng quát của B
- ✓B: Là trường hợp đặc biệt của A



- ✓ A: Là trường hợp tổng quát của B và C
- ✓ B, C: Là trường hợp đặc biệt của A

2. 1CÁC KÝ HIỆU

Ví dụ 1: LỚP NGƯỜI



2.1 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
{
        // …
}
```

2.1 KHAI BÁO

```
class TênLớpCha
{
    Thuộc tính và phương thức của lớp cha
}
class TênLớpDẫnXuất extends TênLớpCha
{
    Thuộc tính và phương thức bổ sung của lớp dẫn xuất
}
```

Ví dụ

```
class COSO
{
    protected kiểu data1;
    protected kiểu data2;
      public void Method1()  {    }
      public void Method2()  {    }
}
```

```
class DANXUAT extends
COSO
{
    private kiểu data3;          {   }
    public void  Method4()       {   }
    public void  Method5()
}
```

Ví du:

```
public  class Nguoi
{           protected String ten;
            protected String  diachi;
            public void nhapten() {    // Thuc hien cong viec }
            public void nhapdiachi(){  //Thuc hien cong viec  }
}
public class Sinhvien extends Nguoi
{           private float diemmon1;
            private float diemmon2;
            public Sinhvien(); //ham khoi tao
            public void nhapdiem(){ //thuc hien cong viec}
            public float tinhtong(){//thuc hien cong viec}
}
public class Nhanvien extends Nguoi
{           private long luong;
            private String  chucvu;
            public Nhanvien(); //Ham khoi tao
            public void nhapchucvu(){ //thuc hien cong viec}
            public float tinhluong(){//thuc hien cong viec}
}
```

2.1 GỌI PHIÊN BẢN PHƯƠNG THỨC CỦA LỚP CHA

- Đôi khi, tại một lớp con, ta cài đè một hành vi của lớp cha, nhưng ta không muốn thay thế hoàn toàn mà chỉ muốn bổ sung một số chi tiết.
- Tóm lại, từ trong phiên bản cài đè tại lớp con, ta muốn gọi đến chính phương thức đó của lớp cha, ta phải làm như thế nào?
- **Từ khóa super trong java** là một biến tham chiếu được sử dụng để tham chiếu trực tiếp đến đối tượng của lớp cha gần nhất.
- Từ khóa super cho phép gọi đến cách thành viên được thừa kế

2.1 GỌI PHIÊN BẢN PHƯƠNG THỨC CỦA LỚP CHA

Ví dụ: Không sử dụng **super**

```
class Vehicle {
    int speed = 50;
}

 public class Bike extends Vehicle {
    int speed = 100;

    void display() {
        System.out.println(speed); //in speed của lớp Bike
    }

    public static void main(String args[]) {
        Bike b = new Bike();
        b.display();
    }
}
```

Kết quả:

```
100
```

Ví dụ: Sử dụng **super**

```
class Vehicle {
    int speed = 50;
}

public class Bike2 extends Vehicle {
    int speed = 100;

    void display() {
        System.out.println(super.speed);//in speed của lớp Vehicle
    }

    public static void main(String args[]) {
        Bike2 b = new Bike2();
        b.display();

    }
}
```

Kết quả:

```
50
```

2.1 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 Sinhvien extends Nguoi
{
    public Sinhvien( ) { }
    public Sinhvien(String s_ten, String s_diachi, float f_dm1, float f_dm2)
    {   this.ten = s_ten;
        this.diachi= s_diachi;
        this.diemmon1 = f_dm1;
        this.diemmon2 = f_dm2;
    }
    public void nhapten( ) {//dinh nghia lai phuong thuc nhap ten}
    public void nhapdiachi(){//dinh nghia lai phuong thuc nhap dia chi}
    public void nhapdiem(){ //thuc hien cong viec}
    public float tinhtong(){//thuc hien cong viec}
}
```

2.1 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.

2.1 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

2.1 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()
        {
                // …
        }
}
```

2.1. BÀI TẬP

Thiết kế chương trình quản lý các đối tượng sau trong một Viện khoa học: nhà khoa học, nhà quản lý và nhân viên phòng thí nghiệm. Một nhà khoa học cũng có thể làm công tác quản lý. Các thành phần dữ liệu của các đối tượng trên:

- ***Nhà khoa học***: họ tên, năm sinh, số bài báo đã công bố, số ngày công trong tháng, bậc lương
- ***Nhà quản lý:*** họ tên, năm sinh, số ngày công trong tháng, bậc lương
- ***Nhân viên phòng thí nghiệm:*** họ tên, năm sinh, lương trong tháng.

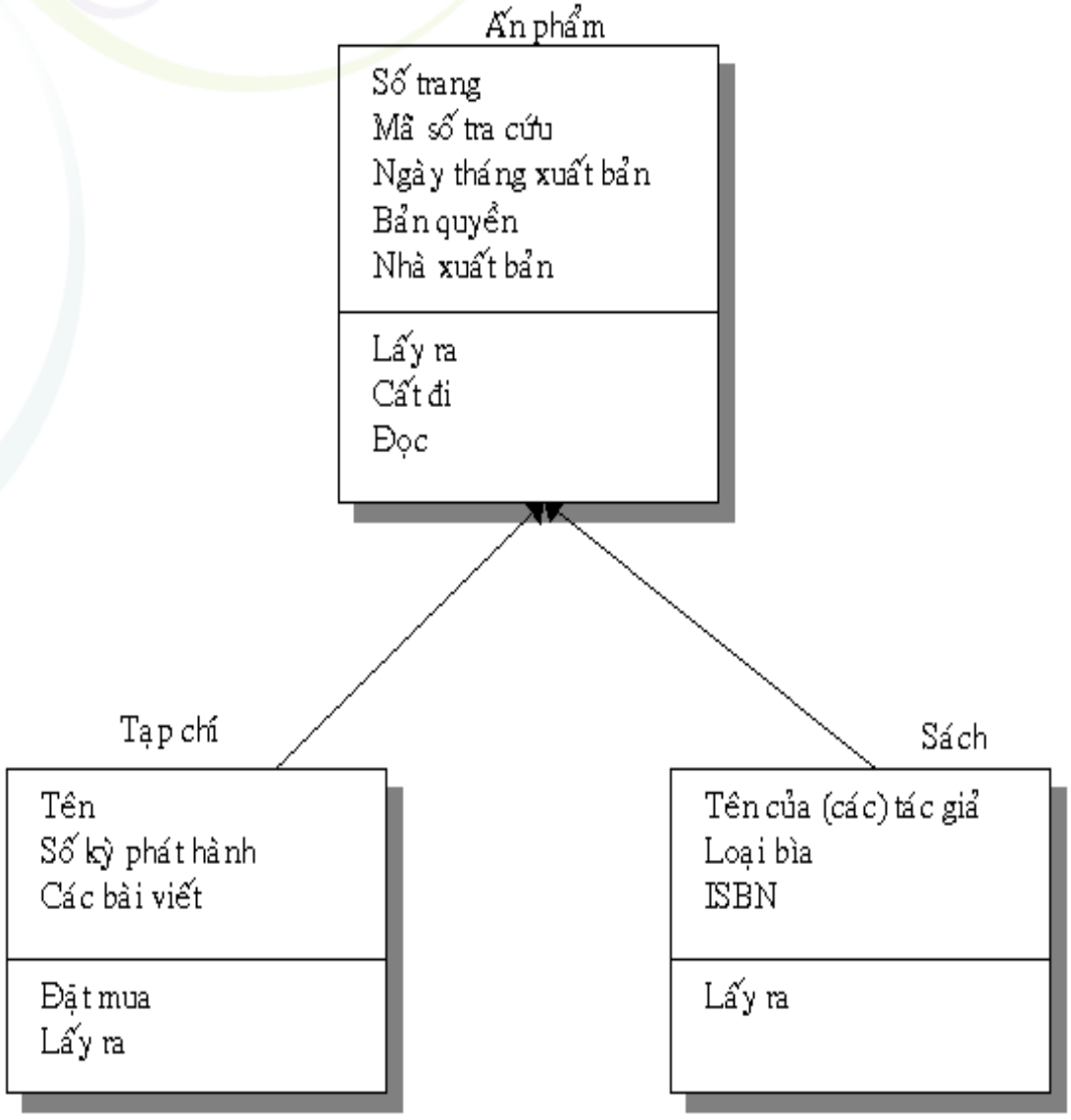
❖ **Thực hiện các yêu cầu sau:**

1. Cài đặt các phương thức thiết lập để nhập liệu, biết rằng nhân viên phòng thí nghiệm lãnh lương khoán, còn lương của nhà khoa học và nhà quản lý bằng số ngày công trong tháng * bậc lương * Lương cơ bản (giả sử = 400,000).
2. Xuất dữ liệu ra màn hình
3. In tổng lương đã chi trả cho từng loại đối tượng

egenera

ĐA HÌNH VÀ TRỪU TƯỢNG

2.2 TÍNH ĐA HÌNH



2.2. VÍ DỤ 1

```
static void Main()
{
    AnPham a = new AnPham();
    a.LayRa();

    TapChi t = new TapChi();
    t.LayRa();

    a = t;
    a.LayRa();
}
```

Tạp Chí ?
Ấn Phẩm ?

2.2 VÍ DỤ 2

Làm thế nào lưu danh sách (mảng) 2 loại ấn phẩm cùng lúc?

Trả lời: Dùng 1 mảng ANPHAM, sau đó tùy vào Người dùng nhập loại nào (TAPCHI hay SACH) thì ta sẽ khởi tạo đối tượng tương ứng

```
static void Main()
{
    AnPham[] ds = new AnPham[100];
    for(int i=0;i<100;i++)
    {
        if( Nhập Loại nào?)
                ds[i] = new TapChi();
        else
                ds[i] = new Sach();
    }
}
```

2.2 KHÁI NIỆM ĐA HÌNH

- **Đa hình trong java (Polymorphism)** là một khái niệm mà chúng ta có thể thực hiện một hành động bằng nhiều cách khác nhau
- Có hai kiểu của đa hình trong java, đó là đa hình lúc biên dịch (compile) và đa hình lúc thực thi (runtime). Chúng ta có thể thực hiện đa hình trong java bằng cách nạp chồng phương thức và ghi đè phương thức.

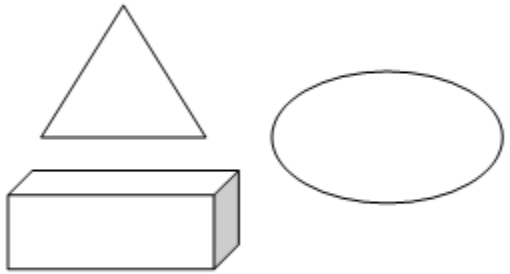
2.2 TÍNH ĐA HÌNH (polymorphism)

Tính đa hình cho phép một phương thức có các tác động khác nhau trên nhiều loại đối tượng khác nhau

Lớp: Hình thể

Các phương thức:
Vẽ
Di chuyển
Khởi tạo

Các lớp con



Hình 1.5: Lớp 'Hình thể' và các lớp con

2. 2 TÍNH ĐA HÌNH (polymorphism)

Ghi đè phương thức

- Phương thức của lớp con có cùng tên và đặc điểm như phương thức của lớp cha (có cùng danh sách tham số và có cùng kiểu trả về).
- Khi phương thức ghi đè được triệu hồi từ lớp con, ta luôn có phiên bản phương thức ở lớp con được thực hiện. Xét chương trình Override sau:

2.2 TÍNH ĐA HÌNH (polymorphism)

```
package chuong5;
class A {
      int i, j;
      A(int a, int b) {
          i  = a;
          j  = b;
      }
     // hiển thị i và j
     void show() {
        System.out.println("i and j: " + i + " " + j);
      }
}//End Class A.
class B extends A {
            int k;
      B(int a, int b, int c) {
          super(a, b);
          this.k = c;
      }
// hiển thị k – phương thức này ghi đè
void show() {
            System.out.println("k: " + k);
}
}
public class Test {
public static void main(String args[]) {
            B subOb = new B(1, 2, 3);
             // this calls show() in B
            subOb.show();
   }
}
```

Kết quả là: k:3

2.2 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 {
    // 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
```

2.2 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
```

2.2 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.

2.2 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
```

2.2 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
```

2.2 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.

2.2 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
```

2.2 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";
}
```

2.2 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.

2.2 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

2.2 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
```

2.2 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();
}
```

2.2 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 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
```

2.2 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
```

2.2 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

2.2 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
```

2.2 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
```

2.2 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
```

2.2 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
```

2.2 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
```

2.2 LỚP TRỪU TƯỢNG (ABSTRACT)

- Lớp trừu tượng là một dạng lớp đặc biệt, trong đó có phương thức chỉ được khai báo ở dạng khuôn mẫu (template) mà không được cài đặt chi tiết. Việc cài đặt chi tiết các phương thức chỉ được thực hiện ở các lớp con kế thừa lớp trừu tượng đó.
- Lớp trừu tượng được sử dụng khi muốn định nghĩa một lớp mà không thể biết và định nghĩa ngay được vài phương thức của nó.
- Lớp trừu tượng chỉ thiết lập cơ sở cho các lớp kế thừa => bên trong nó sẽ **không có bất kỳ cài đặt** nào khác
- Cú pháp khai báo lớp trừu tượng:

  [public]abstract class <Tên lớp>

- Lớp trừu tượng là công cụ để ta định nghĩa các hành vi và các thuộc tính mà một nhóm lớp nào đó bắt buộc phải có. Bởi vì các lớp con của nó bắt buộc phải hiện thực các phương thức trừu tượng của nó. **Ví dụ:** Lớp máy nghe nhạc bắt buộc phải có các hành vi chung đó là: Play, Stop, Pause, Resume, Forward, Backward.

2.2 LỚP TRỪU TƯỢNG (ABSTRACT)

Lưu ý:

Lớp trừu tượng cũng có thể kế thừa một lớp khác, nhưng lớp cha cũng phải là một lớp trừu tượng. (Khai báo kế thừa thông qua từ khoá extends như khai báo kế thừa thông thường).

Khai báo phương thức của lớp trừu tượng

[public] abstract <kiểu dữ liệu trả về> <Tên phương thức> ([<các tham số>]) [throws
<các ngoại lệ>];

2.2 LỚP TRỪU TƯỢNG (ABSTRACT)

- Tính chất: tính chất của một phương thức trừu tượng của lớp trừu tượng luôn là public. Nếu không khai báo tường minh thì giá trị mặc định cũng là public.
- Kiểu dữ liệu trả về: có thể là các kiểu cơ bản của Java, cũng có thể là kiểu do người dùng tự định nghĩa (kiểu đối tượng).
- Tên phương thức: tuân thủ theo quy tắc đặt tên phương thức của lớp
- Các tham số: nếu có thì mỗi tham số được xác định bằng một cặp <kiểu tham số> <tên tham số>. Các tham số được phân cách nhau bởi dấu phẩy.
- Các ngoại lệ: nếu có thì mỗi ngoại lệ được phân cách nhau bởi dấu phẩy.

2.2 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.

2.2 LỚP TRỪU TƯỢNG (ABSTRACT)

Ví dụ: Lớp trừu tượng và phương thức trừu tượng

```
// lop truu tuong Bike
abstract class Bike{
  abstract void run();  // phuong thuc truu tuong voi tu khoa abstract
}

// lop Honda4 ke thua lop truu tuong Bike
class Honda4 extends Bike{
void run(){
  System.out.println("Dang chay mot cach an toan..");
  }

// phuong thuc main()
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

2.2 LỚP TRỪU TƯỢNG (ABSTRACT)

Ví dụ: Kế thừa lớp Abstract trong Java

```
abstract class Bank{
 abstract int getRateOfInterest();
}
class SBI extends Bank{
 int getRateOfInterest(){return 7;
} // bat buoc phai cung cap trinh trien khai cua getRateOfInterest }

class PNB extends Bank{
 int getRateOfInterest(){return 8;}
// bat buoc phai cung cap trinh trien khai cua getRateOfInterest }
```

2.2 LỚP TRỪU TƯỢNG (ABSTRACT)

Ví dụ: Kế thừa lớp Abstract trong Java

```
class TestBank{
 public static void main(String args[]){
// Tao mot doi tuong SBI moi
Bank b=new SBI(); //Neu doi tuong la PNB, phuong thuc cua PNB se duoc trieu ho
 int interest=b.getRateOfInterest(); //Trieu hoi phuong thuc cua SBI
 System.out.println("Ti le lai suat la: "+interest+" %");
}}
```

2.2 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.
            // …
        }
}
```

2.2 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;*

egenera

GIAO DIỆN(Interface)

2.3 GIAO DIỆN

- ✓Giao diện là một dạng của lớp trừu tượng được sử dụng với mục đích hỗ trợ tính đa hình
- ✓Trong giao diện không có bất cứ một cài đặt nào, Chỉ có các nguyên mẫu phương thức, thuộc tính, chỉ mục, được khai báo trong giao diện.
- ✓Tất cả các thành phần khai báo trong giao diện mặc định là public (nên không có từ khóa về mức độ truy cập trong khai báo các thuộc tính và phương thức)
- ✓Khi một lớp kế thừa một giao diện ta nói rằng lớp đó thực thi (Implement) giao diện

2.3 GIAO DIỆN

✓Nhằm tránh những nhập nhằng của tính chất đa kế thừa của C++, Java không cho phép kế thừa trực tiếp từ nhiều hơn một lớp cha. Nghĩa là Java không cho phép đa kế thừa trực tiếp, nhưng cho phép cài đặt nhiều giao tiếp (Interface) để có thể thừa hưởng thêm các thuộc tính và phương thức của các giao tiếp đó.

2.3 GIAO DIỆN

Tách biệt giữa giao tiếp và cài đặt cụ thể

Làm cái gì? **interface**

Làm bằng cách nào? **Implementation**

2.3 GIAO DIỆN

2. Khai báo giao diện

[MứcĐộTruyCập] Interface TênGiaoDiện[extends <danh sách giao tiếp>]

{

//Nội dung giao diện

}

Trong đó:

- *[MứcĐộTruyCập] : thường là public; nếu không khai báo tường minh thì mặc định là public*
- *[danh sách giao tiếp] : danh sách các Interface cha đã được định nghĩa để kế thừa, các Interface được phân cách nhau bởi dấu phẩy*

2.3 GIAO DIỆN (interface)

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.

2.3 GIAO DIỆN

✓*Khai báo phương thức của giao tiếp*

[public] <kiểu giá trị trả về> <Tên phương thức> ([<các tham số>]) [throws <danh sách ngoại lệ>];

- Tính chất: tính chất của một thuộc tính hay phương thức của giao tiếp luôn là public. Nếu không khai báo tường minh thì giá trị mặc định cũng là public. Đối với thuộc tính, thì luôn phải thêm là hằng (final) và tĩnh (static).
- Kiểu giá trị trả về: có thể là các kiểu cơ bản của Java, cũng có thể là kiểu do người dùng tự định nghĩa (kiểu đối tượng).
- Tên phương thức: tuân thủ theo quy tắc đặt tên phương thức của lớp
- Các tham số: nếu có thì mỗi tham số được xác định bằng một cặp <kiểu tham số> <tên tham số>. Các tham số được phân cách nhau bởi dấu phẩy.
- Các ngoại lệ: nếu có thì mỗi ngoại lệ được phân cách nhau bởi dấu phẩy.

2.3 GIAO DIỆN

Lưu ý:

- Các phương thức của giao tiếp chỉ được khai báo dưới dạng mẫu mà không có cài đặt chi tiết (có dấu ; ngay sau khai báo và không có phần cài đặt trong dấu { }).
- Phần cài đặt chi tiết của các phương thức chỉ được thực hiện trong các lớp sử dụng giao tiếp đó.
- Các thuộc tính của giao tiếp luôn có tính chất là final, static và public. Do đó, cần gán giá trị khởi đầu ngay khi khai báo thuộc tính của giao tiếp.

2.3 GIAO DIỆN

Phân Biệt lớp Abstract và Interface

Lớp trừu tượng	Interface
Lớp trừu tượng có thể có các phương thức abstract và non-abstract	Interface chỉ có thể có phương thức abstract
Lớp trừu tượng không hỗ trợ đa kế thừa	Interface hỗ trợ **đa kế thừa**
Lớp trừu tượng có thể có các biến **final, non-final, static và non-static**	Interface chỉ có các biến **static và final**
Lớp trừu tượng có thể có phương thức static, phương thức main và constructor	Interface không thể có phương thức static, main hoặc constructor.
Từ khóa abstract được sử dụng để khai báo lớp trừu tượng	Từ khóa interface được sử dụng để khai báo Interface
Lớp trừu tượng có thể cung cấp trình triển khai của Interface	Interface không cung cấp trình triển khai cụ thể của lớp abstract
Ví dụ: public abstract class Shape{ public abstract void draw(); }	Ví dụ: public interface Drawable{ void draw(); }

2.3 VÍ DỤ GIAO DIỆN

Ví dụ: *Interface đơn giản*

```
interface printable{
    void print();
}
class A6 implements printable{
    public void print(){
            System.out.println("Hello");
    }
    public static void main(String args[]){
            A6 obj = new A6(); obj.print();
    }
 }
```

2.3 VÍ DỤ GIAO DIỆN

Ví dụ: Đa kế thừa trong Java bởi Interface

```
interface Printable{
void print();
}

interface Showable{
void show();
}

class A7 implements Printable,Showable{

public void print(){System.out.println("Hello");}
public void show(){System.out.println("Welcome");}

public static void main(String args[]){
A7 obj = new A7();
obj.print();
obj.show();
 }
}
```

2.3 VÍ DỤ GIAO DIỆN

Ví dụ: **Câu hỏi**: Đa kế thừa không được hỗ trợ thông qua lớp trong Java nhưng là có thể bởi Interface, tại sao?

Như đa thảo luận trong chương về tính kế thừa, đa kế thừa không được hỗ trợ thông qua lớp. Nhưng nó được hỗ trợ bởi Interface bởi vì không có tính lưỡng nghĩa khi trình triển khai được cung cấp bởi lớp Implementation. Ví dụ:

```
interface Printable{
void print();
}
interface Showable{
void print();
}

class TestInterface1 implements Printable,Showable{
public void print(){System.out.println("Hello");}
public static void main(String args[]){
TestInterface1 obj = new TestInterface1();
obj.print();
 }
}
```

2.3 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); }
}
```

2.3 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 ",".

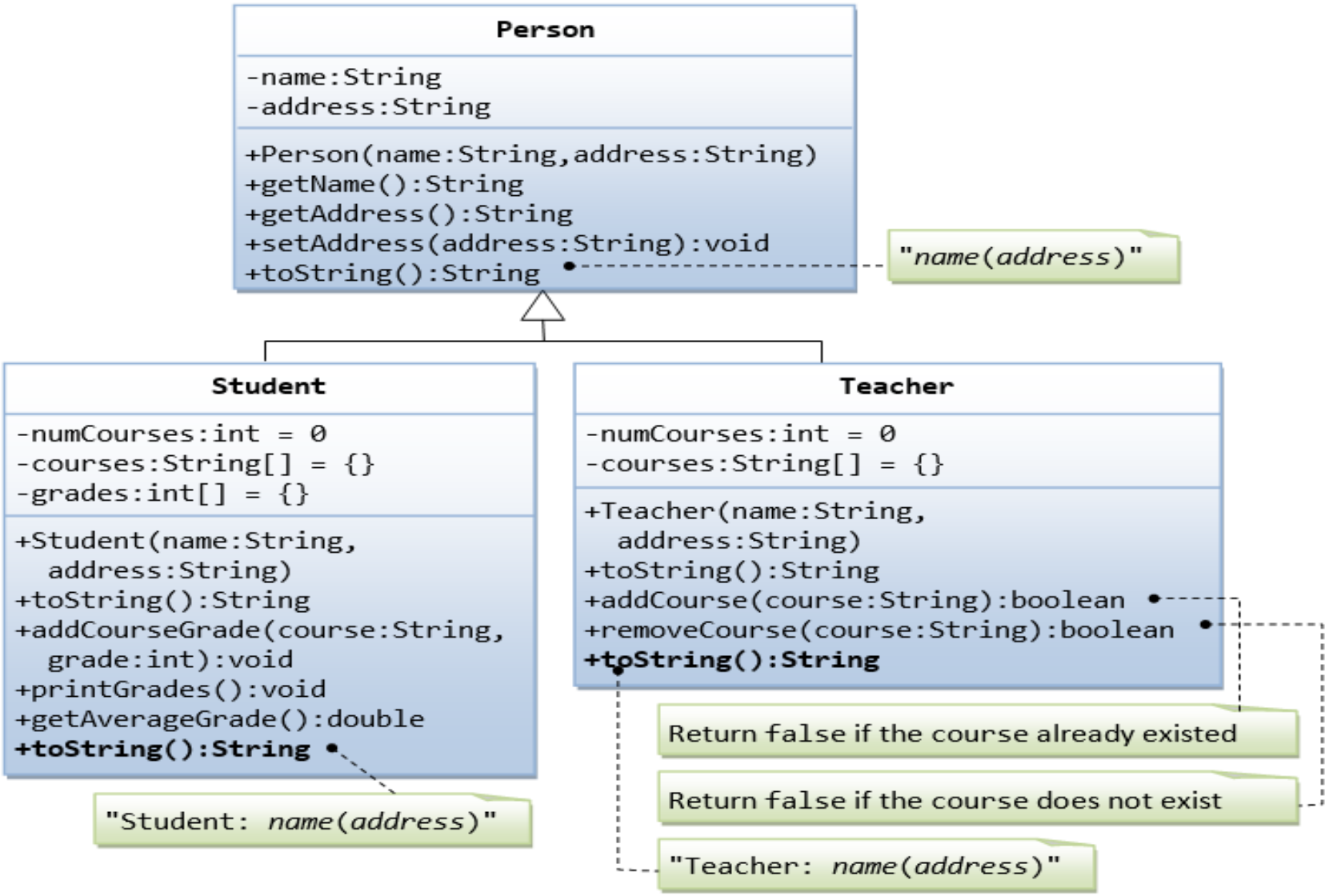
Khai báo như sau:

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

BÀI TẬP 1

Giả sử chúng ta phải mô hình hóa sinh viên và giáo viên trong ứng dụng của chúng ta. Chúng ta có thể định nghĩa một siêu lớp được gọi là Người để lưu trữ các thuộc tính phổ biến như tên và địa chỉ và phân lớp Học sinh và Giáo viên cho các thuộc tính cụ thể của chúng. Đối với sinh viên, chúng ta cần duy trì các khóa học đã học và các điểm tương ứng; thêm một khóa học với lớp, in tất cả các khóa học đã thực hiện và tính trung bình. Giả sử rằng một sinh viên mất không quá 30 khóa học cho toàn bộ chương trình. Đối với giáo viên, chúng ta cần phải duy trì các khóa học được giảng dạy hiện nay và có thể thêm hoặc xóa một khóa học được giảng dạy. Giả sử rằng một giáo viên dạy không quá 5 khóa học đồng thời.

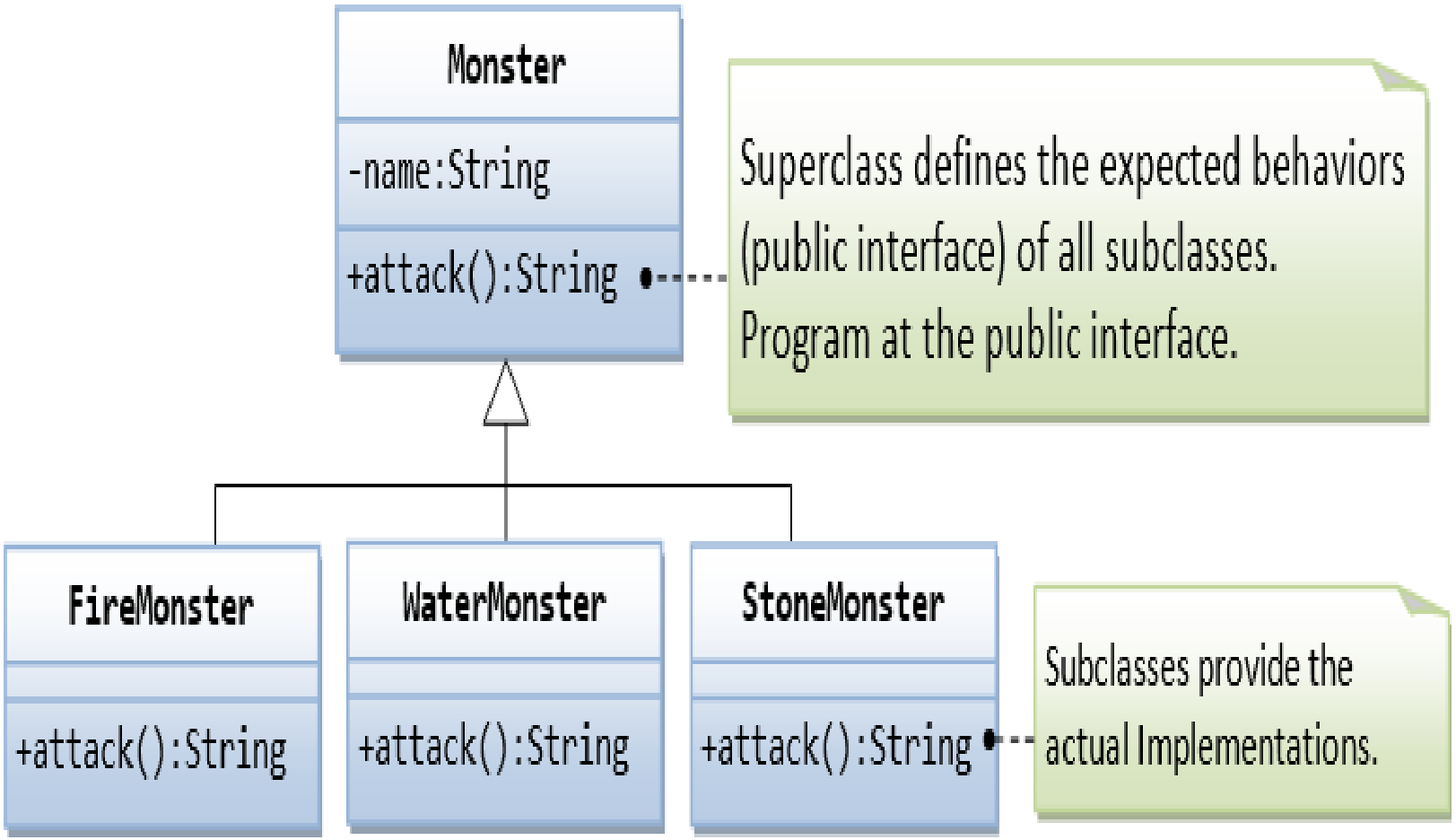
BÀI TẬP 1



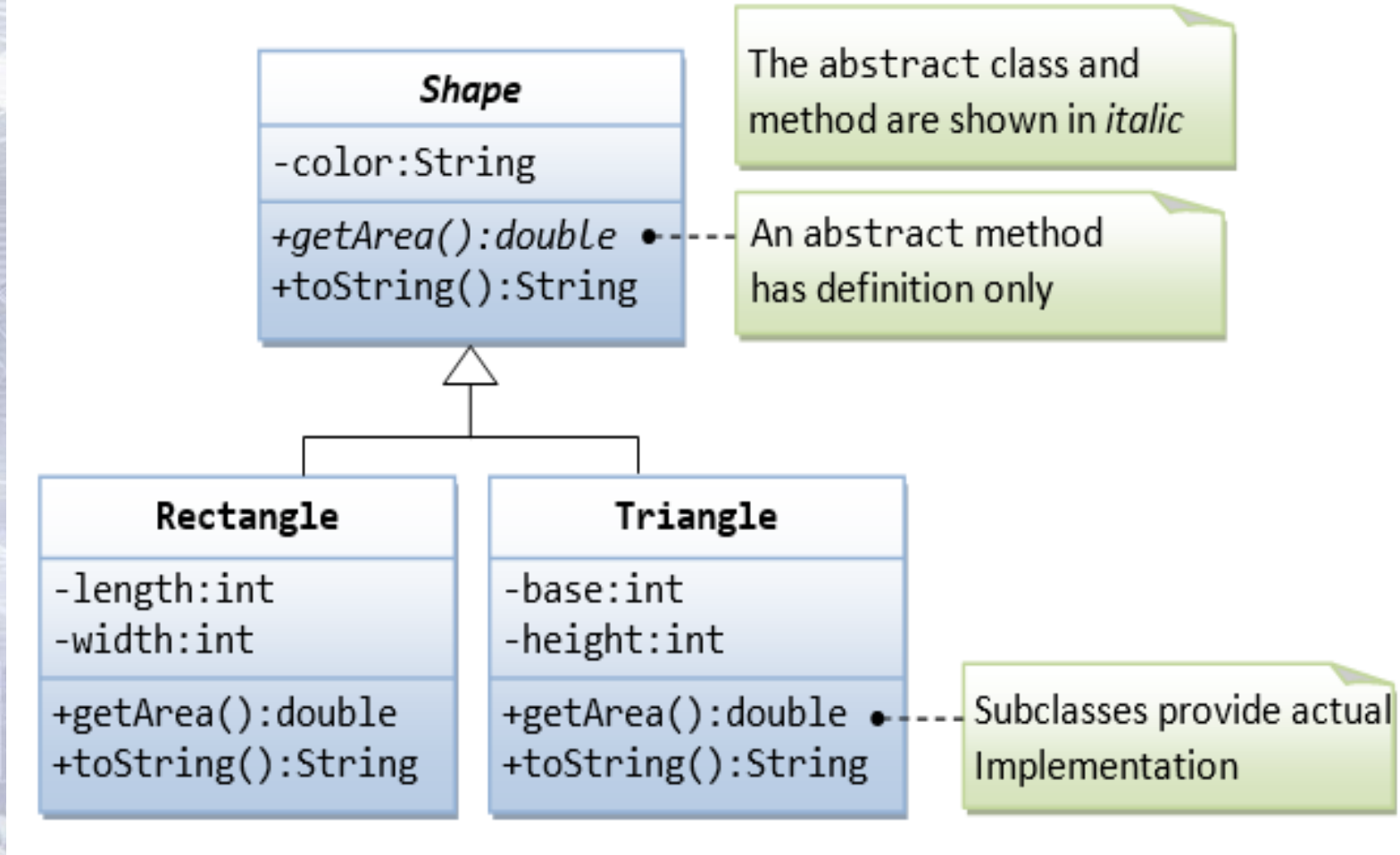
BÀI TẬP 2

Trong ứng dụng trò chơi của chúng tôi, chúng tôi có nhiều loại quái vật có thể tấn công. Chúng ta sẽ thiết kế một superclass mang tên Monster và xác định phương thức attack () trong superclass. Các lớp con sẽ cung cấp thực hiện thực tế của chúng. Trong chương trình chính, chúng ta khai báo các thể hiện của superclass, được thay thế bằng subclass thực; và gọi phương thức được định nghĩa trong lớp cha.

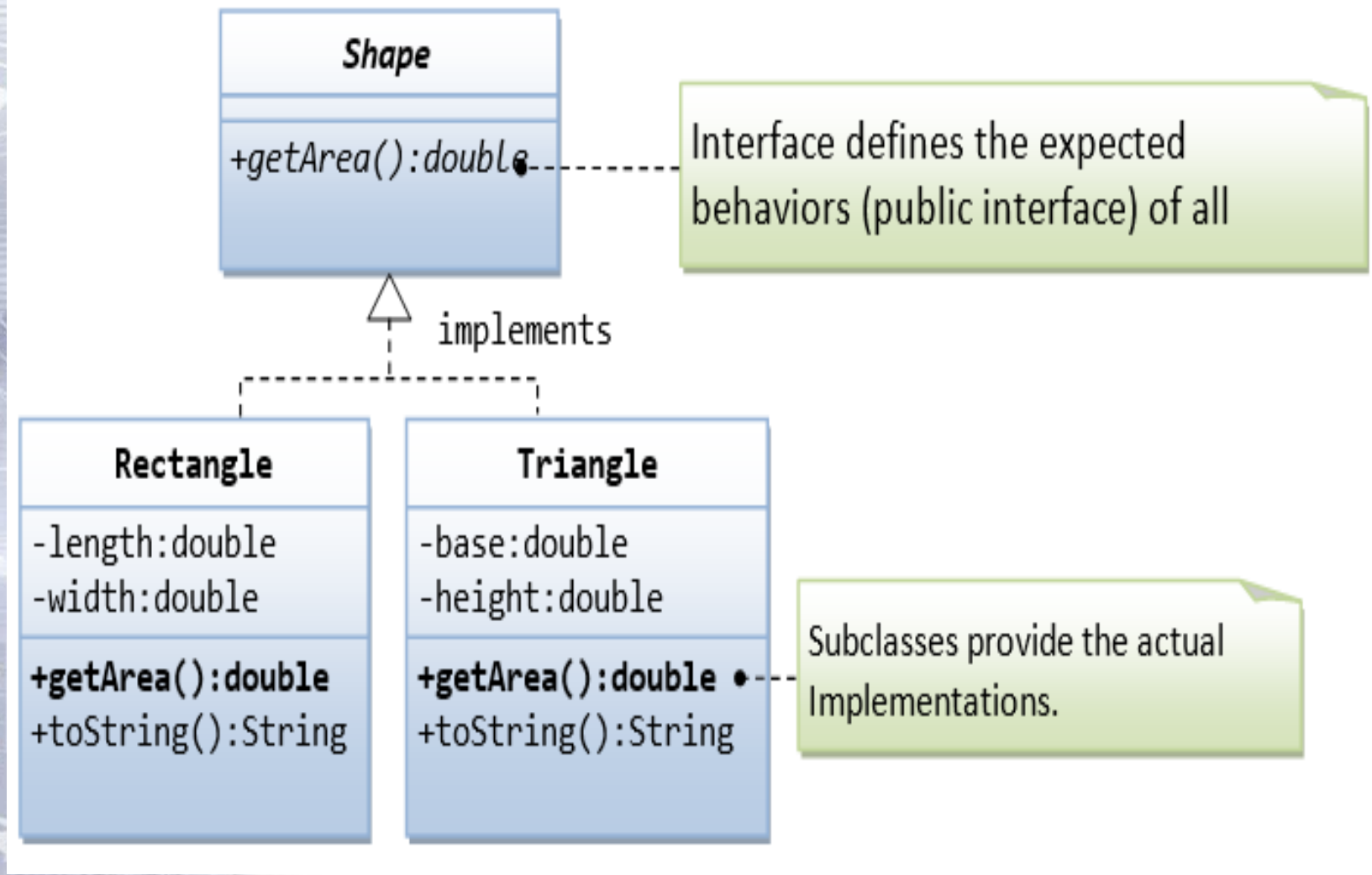
BÀI TẬP 2



BÀI TẬP 3 Abstract & Interface



BÀI TẬP 3 Abstract & Interface



XỬ LÝ NGOẠI LỆ
(EXCEPTION)

NỘI DUNG TRÌNH BÀY

Xử lý lỗi và ngoại lệ

Khối try/catch/finally

Các lớp ngoại lệ

Xây dựng lớp ngoại lệ

Lan truyền ngoại lệ

Tung lại ngoại lệ

Bài tập

XỬ LÝ LỖI VÀ NGOẠI LỆ

Trong một số ngôn ngữ như C, việc xử lý lỗi thường được cài đặt ngay tại các bước thực hiện của chương trình. Các hàm sẽ trả về một cấu trúc lỗi khi gặp lỗi.

Ví dụ: Tìm kiếm phần tử trong một danh sách

```
ErrorStruct  error = new ErrorStruct();
TableEntry entry = lookup("Marianna", employee, error);
if (entry == null)
{
    return error;
}
```

XỬ LÝ LỖI VÀ NGOẠI LỆ

⇒Mã lệnh và mã xử lý lỗi nằm xen kẽ khiến lập trình viên khó theo dõi được thuật toán chính của chương trình.

⇒Khi một lỗi xảy ra tại hàm A, tất cả các lời gọi hàm lồng nhau đến A đều phải xử lý lỗi mà A trả về.

XỬ LÝ LỖI VÀ NGOẠI LỆ

Trong Java, việc xử lý lỗi có thể được cài đặt trong một nhánh độc lập với nhánh chính của chương trình.

Lỗi được coi như những trường hợp ngoại lệ (exceptional conditions). Chúng được bắt/ném (catch and throw) khi có lỗi xảy ra.

=> Một trường hợp lỗi sẽ chỉ được xử lý tại nơi cần xử lý.

=> Mã chính của chương trình sáng sủa, đúng với thiết kế thuật toán.

VÍ DỤ 1

```
import java.awt.Point;

public class MyArray
{
    public static void main(String[ ] args) {
        System.out.println("Goi phuong thuc methodeX()");
        methodeX();
        System.out.println("Chuong trinh ket thuc binh thuong");
    }

    public static void methodeX()  {
        Point[ ] pts = new Point[10];
        for(int i = 0; i < pts.length; i++)  {
            pts[i].x = i;
            pts[i].y = i+1;
        }
    }
}
```

KẾT QUẢ THỰC THI VÍ DỤ 1

```
Goi phuong thuc methodeX()
Exception in thread "main" java.lang.NullPointerException
        at MyArray.methodeX(MyArray.java:14)
        at MyArray.main(MyArray.java:7)
```

Giải thích: Hệ thống đã tung ra một exception thuộc lớp NullPointerException khi gặp lỗi. Sau đó chương trình kết thúc.

VÍ DỤ 2

```
public class MyDivision {
    public static void main(String[ ] args)  {
        System.out.println("Goi phuong thuc A()");
        A();
        System.out.println("Chuong trinh ket thuc binh thuong");
    }
    public static void A()  {
        B();
    }
    public static void B()  {
        C();
    }
    public static void C()  {
        float a = 2/0;
    }
}
```

KẾT QUẢ THỰC THI VÍ DỤ 2

```
Goi phuong thuc A()
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at MyDivision.C(MyDivision.java:14)
        at MyDivision.B(MyDivision.java:11)
        at MyDivision.A(MyDivision.java:8)
        at MyDivision.main(MyDivision.java:4)
```

Giải thích: Phương thức A() gọi B(), B() gọi C(), C() gây ra lỗi chia cho 0 và hệ thống "ném" ra một exception thuộc lớp ArithmeticException. Sau đó chương trình kết thúc.

NGOẠI LỆ

Khi một phương thức gặp lỗi nào đó, ví dụ như chia không, vượt kích thước mảng, mở file chưa tồn tại… thì các ngoại lệ sẽ được ném ra. Chương trình dừng lại ngay lập tức, toàn bộ phần mã phía sau sẽ không được thực thi.

Java hỗ trợ cách thức để xử lý ngoại lệ (exception handling) tuỳ theo nhu cầu của chương trình.

XỬ LÝ NGOẠI LỆ

Khối try/catch

Đặt đoạn mã có khả năng xảy ra ngoại lệ trong khối **try**

Đặt đoạn mã xử lý ngoại lệ trong khối **catch**

Khi xảy ra ngoại lệ trong khối try, các câu lệnh trong khối catch sẽ được thực hiện tuỳ vào kiểu của ngoại lệ.

Sau khi thực hiện xong khối catch, điều khiển sẽ được trả lại cho chương trình.

KHỐI TRY/CATCH

Ví dụ 1:

```
try
{
      methodeX();
      System.out.println("Cau lenh ngay sau methodX()");
}
catch (NullPointerException e)
{
      System.out.println("Co loi trong khoi try");
}
System.out.println("Cau lenh sau try/catch");
```

KHỐI TRY/CATCH

Ví dụ 2:

```
try {
        A();
} catch (Exception e) {
        System.out.println("Co loi trong A()");
}
```

Ví dụ 3:

```
try {
        x = System.in.read();
        System.out.println("x = " + x);
} catch (IOException e) {
        System.out.println("Error: " + e.getMessage());
}
```

KHỐI TRY/CATCH

Ví dụ 4:

```
try
{
      String s = buff.readLine();
      int a = Integer.parseInt(s);
      x[i++] = a;
} catch (IOException e) {
      System.out.println("Error IO: " + e.getMessage());
} catch (NumberFormatException e) {
      System.out.println("Error Format: " + e.getMessage());
} catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Error Index: " + e.getMessage());
}
```

KHỐI FINALLY

Khi một ngoại lệ xảy ra, chương trình dừng lại, một số công việc "dọn dẹp" có thể sẽ không được thực hiện (ví dụ như đóng file).

Khối finally đảm bảo rằng các câu lệnh trong đó luôn được thực hiện, kể cả khi ngoại lệ xảy ra.

```
try
{
   doSomething();  // phương thức này có thể gây ra ngoại lệ
} finally {
   cleanup();
}
```

TÓM TẮT VỀ XỬ LÝ NGOẠI LỆ

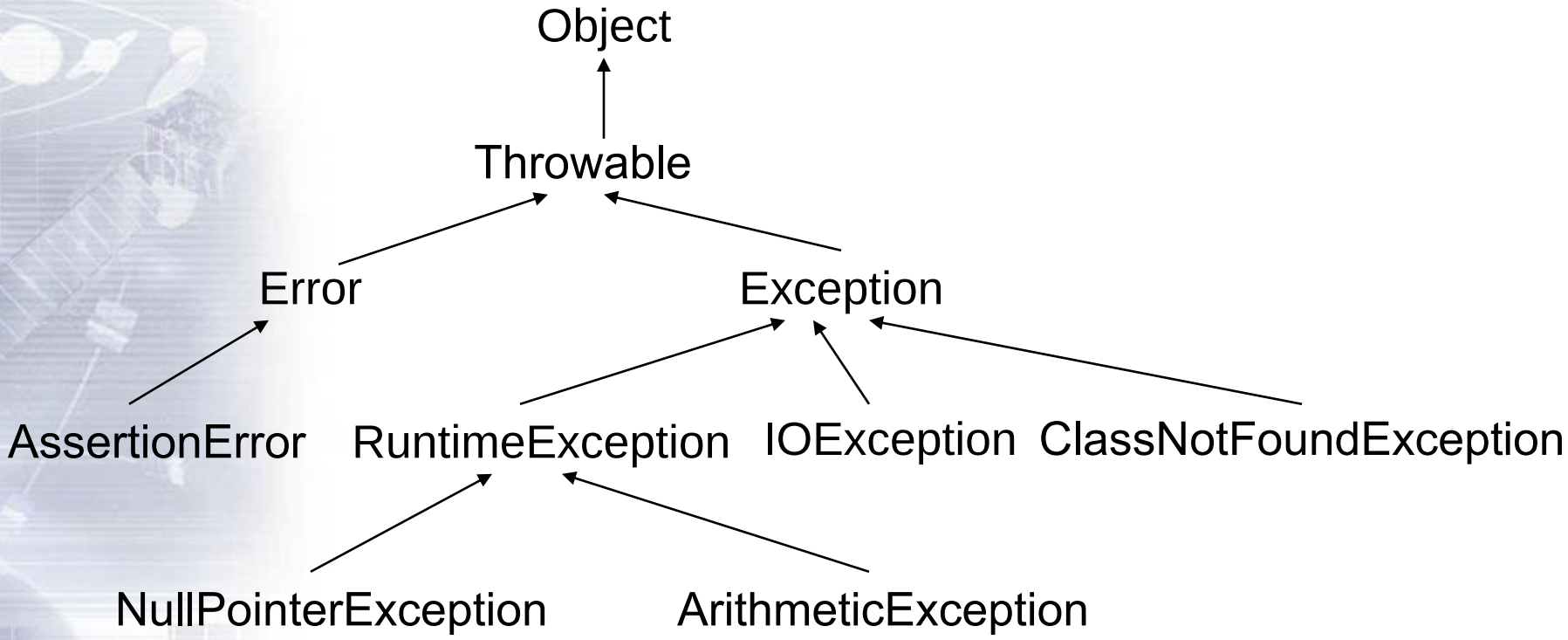
Các ngoại lệ xảy ra khi gặp lỗi.

Có thể bắt và xử lý các ngoại lệ bằng cách sử dụng khối try/catch. Nếu không chương trình sẽ kết thúc ngay (với ứng dụng console) hoặc tiếp tục tồn tại (với ứng dụng GUI).

Khi bắt ngoại lệ, phải biết rõ kiểu ngoại lệ cần bắt. Có thể dùng kiểu cha Exception.

Để chắc chắn việc "dọn dẹp" luôn được thực hiện, dùng khối finally. Có thể kết hợp try/catch/finally.

MỘT SỐ LỚP NGOẠI LỆ



MỘT SỐ LỚP NGOẠI LỆ

Lớp Throwable

Có một biến String để lưu thông tin chi tiết về ngoại lệ đã xảy ra

Một số phương thức cơ bản

Throwable(String s); // Tạo một ngoại lệ có tên là s.

String getMessage(); // Lấy thông tin về ngoại lệ

void printStackTrace(); // In ra tất cả các thông tin liên quan đến ngoại lệ

MỘT SỐ LỚP NGOẠI LỆ

Lớp Exception

Có nhiều ngoại lệ thuộc lớp con của Exception.

Người dùng có thể tạo ra các ngoại lệ kế thừa từ Exception.

Lớp Error

Chỉ những lỗi nghiêm trọng và không dự đoán trước được như ThreadDead, LinkageError, VirtualMachineError...

Các ngoại lệ kiểu Error ít được xử lý.

MỘT SỐ LỚP NGOẠI LỆ

RuntimeException: Chỉ các ngoại lệ có thể xảy ra khi JVM thực thi chương trình

NullPointException: con trỏ null

OutOfMemoryException: hết bộ nhớ

ArithmeticException: lỗi toán học, lỗi chia không…

ClassCastException: lỗi ép kiểu

ArrayIndexOutOfBoundsException: vượt quá chỉ số mảng

…

HAI LOẠI NGOẠI LỆ

Ngoại lệ unchecked

Là các ngoại lệ không bắt buộc phải được kiểm tra.

Gồm RuntimeException, Error và các lớp con của chúng.

Ngoại lệ checked

Là các ngoại lệ bắt buộc phải được kiểm tra.

Gồm các ngoại lệ còn lại.

CHÚ Ý VỚI NGOẠI LỆ CHECKED

Giả sử method1 gọi method2 và method2 là phương thức có khả năng ném ngoại lệ kiểu checked, lúc đó:

hoặc method2 phải nằm trong khối try/catch.

hoặc phải khai báo method1 có khả năng ném (throws) ngoại lệ.

VÍ DỤ (NGOẠI LỆ IOException)

Cách 1: try/catch

```
public static void main(String[] args)
{
   try {
        String s = buff.readLine();
   } catch (IOException e) {
      ...
   }
}
```

Cách 2: Khai báo throws

```
public static void main(String[ ] args) throws IOException
{
   String s = buff.readLine();
}
```

Bài tập

Bài 1: Cài đặt xử lý các ngoại lệ cho chương trình tính thương 2 số bằng giao diện GUI.

Bài 2: Cài đặt xử lý lỗi bằng cách dùng ngoại lệ cho ví dụ ở phần đầu bài.

NGOẠI LỆ DO NGƯỜI DÙNG TẠO

Định nghĩa lớp ngoại lệ

```
// file MyException.java
public class MyException extends Exception
{
        public MyException(String msg)
        {
                super(msg);
        }
}
```

NGOẠI LỆ DO NGƯỜI DÙNG TẠO

Sử dụng ngoại lệ

Khai báo khả năng tung ngoại lệ

```
// file ExampleException.java
public class ExampleException
{
    public void copy(String fileName1, String fileName2)
                                                    throws MyException
    {
        if (fileName1.equals(fileName2))
            throw new MyException("File trung ten"); // tung ngoại lệ

        System.out.println("Copy completed");
    }
```

Tung ngoại lệ

NGOẠI LỆ DO NGƯỜI DÙNG TẠO

Sử dụng ngoại lệ

```
    public static void main(String[] args)
    {
        ExampleException obj = new ExampleException();
        try {
            String a = args[0];
            String b = args[1];
            obj.copy(a,b);
        } catch (MyException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

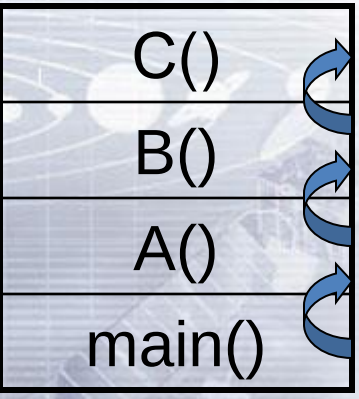
LAN TRUYỀN NGOẠI LỆ

Tình huống

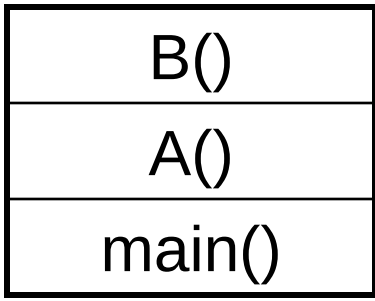
Giả sử trong main() gọi phương thức A(), trong A() gọi B(), trong B() gọi C(). Khi đó một ngăn xếp các phương thức được tạo ra.

Giả sử trong C() xảy ra ngoại lệ.

LAN TRUYỀN NGOẠI LỆ



C() tung ngoại lệ



Nếu C() gặp lỗi và tung ra ngoại lệ nhưng trong C() lại không xử lý ngoại lệ này, thì chỉ còn một nơi có thể xử lý chính là nơi mà C() được gọi, đó là trong phương thức B(). Nếu trong B() cũng không xử lý thì phải xử lý ngoại lệ này trong A()…Quá trình này gọi là lan truyền ngoại lệ.

Nếu đến main() cũng không xử lý ngoại lệ được tung từ C() thì chương trình sẽ phải dừng lại.

NÉM LẠI NGOẠI LỆ

Trong khối catch, ta có thể không xử lý trực tiếp ngoại lệ mà lại ném lại ngoại lệ đó cho nơi khác xử lý.

```
catch (IOException e) {
      throw e;
}
```

- Chú ý: Trong trường hợp trên, phương thức chứa catch phải bắt ngoại lệ hoặc khai báo throws cho ngoại lệ (nếu là loại checked).

CHÚ Ý KHI SỬ DỤNG NGOẠI LỆ

Không nên sử dụng ngoại lệ thay cho các luồng điều khiển trong chương trình.

Ví dụ: Kiểm tra delta trong chương trình giải phương trình bậc 2.

Nên thiết kế và sử dụng ngoại lệ một cách thống nhất cho toàn bộ dự án.

Một số xử lý lỗi bằng ngoại lệ phổ biến là: hết bộ nhớ, vượt quá chỉ số mảng, con trỏ null, chia cho 0, đối số không hợp lệ...

BÀI TẬP

1. *Viết chương trình cho phép tính giá trị của biểu thức:*

$$A = \sqrt{\frac{5x - y}{2x + 7y}}$$

Yêu cầu xử lý các ngoại lệ có thể xảy ra.

2. *Viết chương trình cho phép tạo một mảng 2 chiều cỡ mxn với m,n nhập từ bàn phím. Cài đặt các xử lý ngoại lệ cần thiết.*

BÀI TẬP

3. *Xây dựng lớp ngoại lệ DateException cho các lỗi về ngày tháng.*
4. *Viết chương trình cho phép người dùng nhập vào ngày, tháng năm, nếu thông tin này không hợp lệ sẽ tung ra một ngoại lệ DateException, sau đó thông báo cho người nhập biết và cho phép người dùng nhập lại.*

BÀI TẬP 5

5. *Tìm hiểu lại lớp Candidate đã học (dữ liệu gồm mã thí sinh, tên và điểm thi 3 môn). Điều gì sẽ xảy ra khi tạo một đối tượng thuộc lớp Candidate với dữ liệu đưa vào không hợp lệ ? Cài đặt lớp CandidateException để bắt các lỗi như trên. Yêu cầu khi có lỗi thì sẽ cho biết cả tên và mã thí sinh bị lỗi.*

FAQ