

Buổi 03

Lớp và Đối tượng

(<http://docs.oracle.com/javase/tutorial/java/javaOO/index.html>)

Mục tiêu

Các khái niệm cơ bản về Mô hình lập trình 2-OOP

3-Cách xác định các lớp 4-Cách khai báo / sử dụng một lớp

5-Bổ ngữ phổ biến (một cách để ẩn một số các thành viên trong một lớp học)

6-Quyền truy cập theo dõi đối với các thành viên của một lớp học bằng cách sử dụng bổ ngữ

7-Các phương thức ghi đè trong các lớp con

8-Các lớp lồng nhau

9-Lợi ích của việc triển khai OO:
kế thừa, đa hình

10-Làm việc với các giao diện

11-Làm việc với các phương pháp trừu tượng và
các lớp học

12-Loại Enum

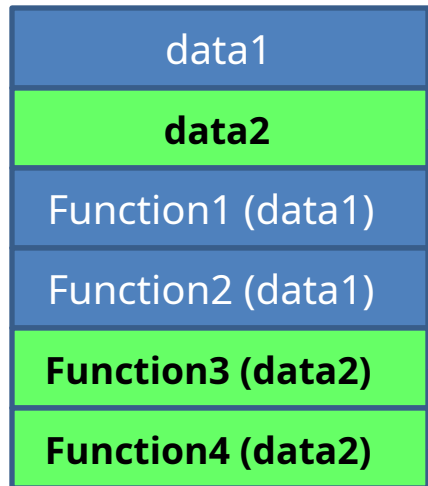
Mô hình lập trình

- Ngôn ngữ lập trình cấp cao (từ 3rd ngôn ngữ thế hệ) được chia thành (Wikipedia):

Mô hình	Sự miêu tả
Mô hình định hướng thủ tục (mệnh lệnh)-POP (3 rd ngôn ngữ thế hệ)	Chương trình = dữ liệu + thuật toán. Mỗi thuật toán được triển khai dưới dạng một hàm (nhóm câu lệnh) và dữ liệu là các tham số của nó (ngôn ngữ C)
Mô hình hướng đối tượng (OOP) (3 rd ngôn ngữ thế hệ)	Chương trình = hành động của một số đối tượng. Đối tượng = dữ liệu + các hành vi. Mỗi hành vi được triển khai dưới dạng một phương thức (C ++, Java, C #,...)
Mô hình chức năng (4 th ngôn ngữ thế hệ)	Ngôn ngữ dành riêng cho miền. Các chức năng cơ bản đã được thực hiện. Chương trình = một tập hợp các hàm (SQL)
Mô hình khai báo / logic (5 th ngôn ngữ thế hệ)	Chương trình = khai báo + quy tắc suy luận (Prolog, CLISP,...)

Mô hình lập trình: POP so với OOP

Chương trình hướng thủ tục



Đối tượng = Dữ liệu + Phương pháp

Các khái niệm cơ bản

- Đóng gói
- Di sản
- Tính đa hình

Cụ thể

phương pháp:
Người xây dựng

Hạng A

{

data1

Hàm1 ()

Function2 ()

}

Hạng B

{

data2

Hàm 3 ()

Hàm4 ()

}

Chung

phương pháp
vì
truy cập một
trường dữ liệu:

Nhập getField ()
void setField (Nhập newValue)

Khái niệm OOP

- Encapsulation
- Inheritance
- Polymorphism

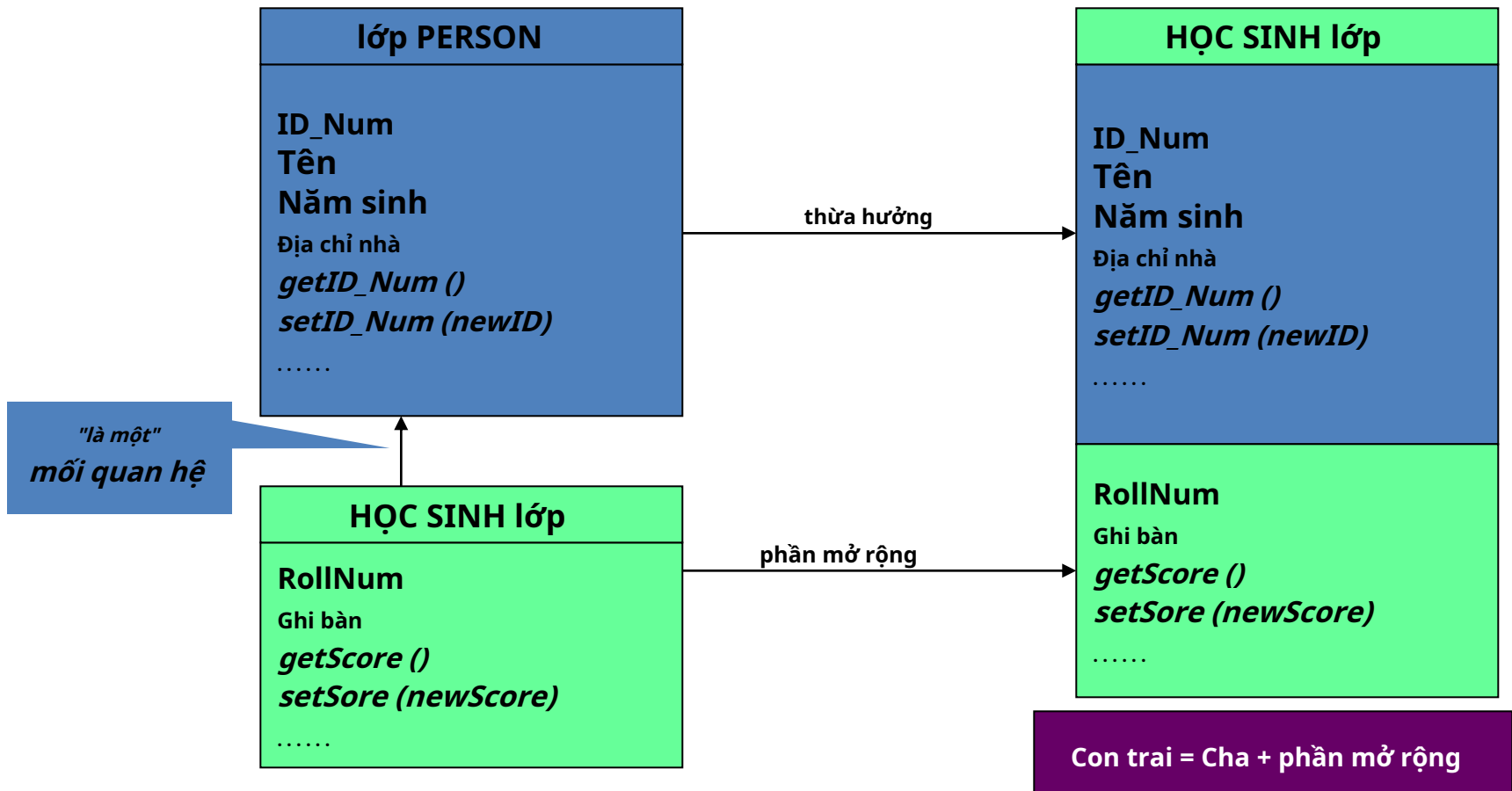
Khái niệm OOP: Đóng gói

Tổng hợp dữ liệu và hành vi.

- Lớp = Dữ liệu (trường / thuộc tính) + Phương thức
- Dữ liệu của một lớp nên được ẩn từ bên ngoài.
- Tất cả các hành vi chỉ nên được truy cập thông qua các phương thức.
- Một phương pháp nên có *điều kiện ranh giới*: Các tham số phải được kiểm tra (sử dụng câu lệnh if) để đảm bảo rằng dữ liệu của một đối tượng luôn hợp lệ.
- **Constructor**: Một phương thức đặc biệt, mã của nó sẽ thực thi khi một đối tượng của lớp này được khởi tạo.

Khái niệm OOP: Thừa kế - 1

Khả năng cho phép một lớp có các thành viên của một lớp đã tồn tại → Mã được sử dụng lại, tiết kiệm thời gian

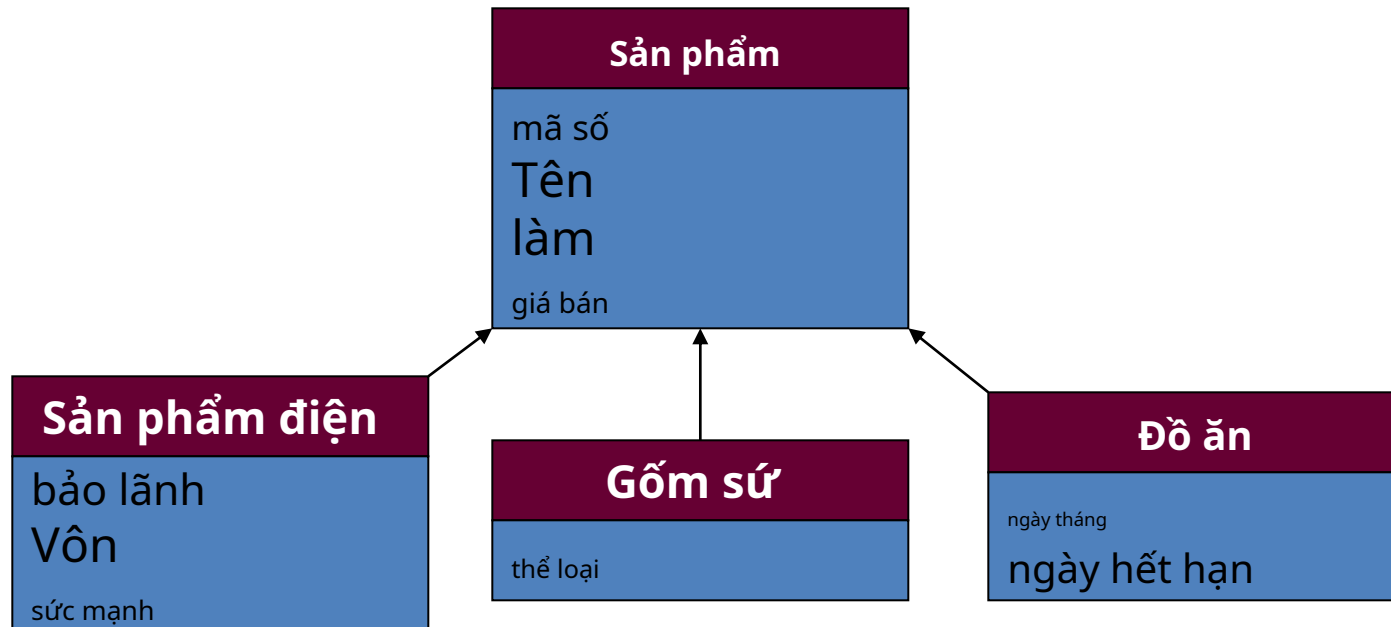


Khái niệm OOP: Thừa kế - 2

Làm thế nào để phát hiện lớp cha?

Tìm giao điểm của các lớp có liên quan.

- Sản phẩm điện <mã, tên, sản xuất, giá cả, đảm bảo, điện áp, điện>
- Sản phẩm gốm <mã, tên, sản xuất, giá cả, gỗ>
- Sản phẩm thực phẩm <mã, tên, sản xuất, giá cả, ngày, hết hạn>



Khái niệm OOP: Polymorphism

Khả năng cho phép nhiều phiên bản của một phương thức dựa trên các kỹ thuật nạp chồng và ghi đè phương thức.

Quá tải: Một lớp có thể có một số phương thức trùng tên nhưng kiểu tham số của chúng khác nhau.

Ghi đè: Một phương thức trong lớp cha có thể bị ghi đè trong các lớp dẫn xuất của nó (phần thân của một phương thức có thể được thay thế trong các lớp dẫn xuất).

Làm thế nào để xác định một lớp học

- Danh từ chính: Lớp
- Danh từ bổ nghĩa cho danh từ chính: Trường
- Các động từ liên quan đến danh từ chính: Phương thức

Phương tiện giao thông có các thuộc tính của nhà sản xuất, năm sản xuất, giá thành, màu sắc

Viết một chương trình Java sẽ cho phép

đầu vào Một Phương tiện giao thông, **output** nó là.

lớp Phương tiện giao thông {

Sợi dây nhà chế tạo;
int năm sản xuất; **chi**
phí gấp đôi;
Màu chuỗi;

Đầu vào phương tiện () {

<mã>

}

void output () {

<mã>

}

}

Khai báo / Sử dụng một lớp Java

[công cộng] **lớp** **Tên lớp** [kéo dài **FatherClass**]{

[sửa đổi] Nhập field1 [= value];

[sửa đổi] Nhập field2 [= value]; //

constructor

[bổ nghĩa] **Tên lớp** (Nhập var1,...){

<mã>

}

[bổ nghĩa] **methodName** (Nhập var1,...){

<mã>

}

.....

}

Bổ ngữ sẽ là giới thiệu sau.

Bao nhiêu các nhà xây dựng nên được thực hiện? → Số lượng cần thiết cách khởi tạo một vật.

Chúng ta sẽ viết gì trong phần thân của constructor? → Họ thường là các mã để khởi tạo giá trị cho các biến mô tả

Xác định cấu trúc

- Các hàm tạo được gọi để tạo các đối tượng từ lớp **bản vẽ thiết kế**.
- Khai báo hàm tạo trông giống như khai báo phương thức - ngoại trừ việc chúng sử dụng **tên của lớp** và có **không có loại trả lại**.
- Trình biên dịch tự động cung cấp một đối số, phương thức khởi tạo mặc định cho bất kỳ lớp nào **không có** các nhà xây dựng.

Phương pháp xác định

- Khai báo phương pháp điển hình:

```
[modifier] ReturnType methodName (params) {  
    <mã>  
}
```

- Chữ ký: dữ liệu giúp xác định điều gì đó
- Chữ ký Phương pháp:
tên + thứ tự của các loại tham số

Truyền đối số một hàm tạo / phương thức

- Java sử dụng cơ chế truyền theo giá trị. Các đối số có thể là:
 - Đối số kiểu dữ liệu ban đầu
 - Đối số kiểu dữ liệu tham chiếu (đối tượng)

Hệ thống quản lý phương tiện

- Mã một lớp có tên **Xe ô tô, Động cơ, Xe tải** được dẫn xuất trực tiếp từ lớp cơ sở **Phương tiện giao thông** vật.

giai cấp công cộng **Phương tiện giao thông** { // siêu cấp

riêng Nhà sản xuất dây;

riêng int năm;

riêng gấp đôi Giá cả;

riêng Sợi dây màu sắc;

công cộng **Phương tiện giao thông**() {}

công cộng **Phương tiện giao thông**(Nhà sản xuất chuỗi,
năm nguyên, chi phí gấp đôi, màu chuỗi) {

 this.manprisurer = nhà sản xuất; this.year
 = năm;

 this.cost = chi phí;

 this.color = màu;

}

```

chuỗi công khai getManufacturer() {
    trả lại nhà sản xuất;}

khoảng trống công cộng thiết lập nhà sản xuất
(Nhà sản xuất chuỗi) {
    this.manosystem = nhà sản xuất;} public int
getYear() {
    năm trở lại;}

khoảng trống công cộng setYear(trong năm) {
    this.year = year;}

công đôi getCost() {
    chi phí trả lại;}

khoảng trống công cộng setCost(chi phí gấp đôi) {
    this.cost = chi phí;}

```

```

chuỗi công khaigetColor() {
    trả lại màu sắc;
}
khoảng trống công cộngsetColor(Màu chuỗi) {
    this.color = màu;
}
chuỗi công khai() {
    trở về
    nhà sản xuất + "\ t" + năm + "\ t" + chi phí + "\ t" + màu r;

}
}

```

Tạo đối tượng

- Lớp cung cấp bản thiết kế cho các đối tượng; bạn tạo một đối tượng từ một lớp.
 - **Điểm p=Mới**Điểm (23, 94);
- **Tuyên bố** có ba phần:
 - **Từ khai**: là tất cả các khai báo biến liên kết tên biến với một kiểu đối tượng.
 - **Thuyết minh**: Từ khóa new là một toán tử Java tạo đối tượng (bộ nhớ được cấp phát).
 - **Khởi tạo**: Toán tử mới được theo sau bởi một cuộc gọi đến một phương thức khởi tạo, khởi tạo đối tượng mới (các giá trị được gán cho các trường).

Loại công trình

Tạo / Sử dụng một đối tượng của một lớp

- **Nhà xây dựng mặc định:** Hàm tạo không có tham số.
- **Hàm tạo tham số:** Hàm tạo với ít nhất một tham số.

- Tạo một đối tượng

ClassName obj1 = new ClassName ();

ClassName obj2 = new ClassName (params);

- Truy cập một trường của đối tượng

object.field

- Gọi một phương thức của một đối tượng
object.method (params)

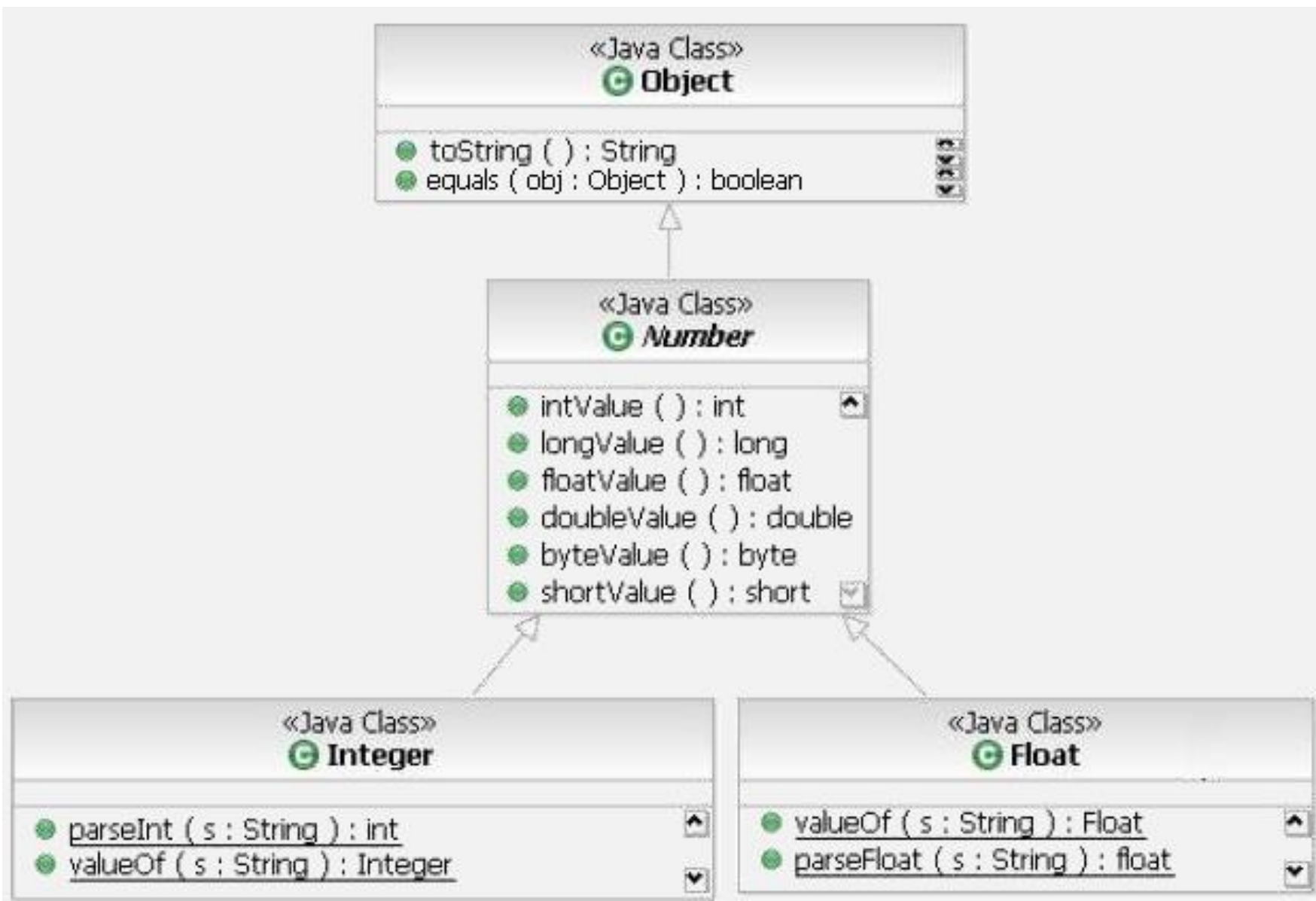
Ghi chú về Trình xây dựng

- Nếu chúng ta không triển khai bất kỳ hàm tạo nào, trình biên dịch sẽ chèn vào lớp một hệ thống mặc định constructor.
- Nếu chúng tôi thực hiện **Một** phương thức khởi tạo, trình biên dịch không **không phải** chèn hàm tạo mặc định.

Lớp đối tượng

- Các **java.lang.Object** lớp là gốc của hệ thống phân cấp lớp. Mọi lớp đều có Đối tượng là lớp cha. Tất cả các đối tượng, bao gồm cả mảng, thực thi các phương thức của lớp này.
- **Các hàm tạo lớp: Object ()** Đây là một cấu trúc đơn.

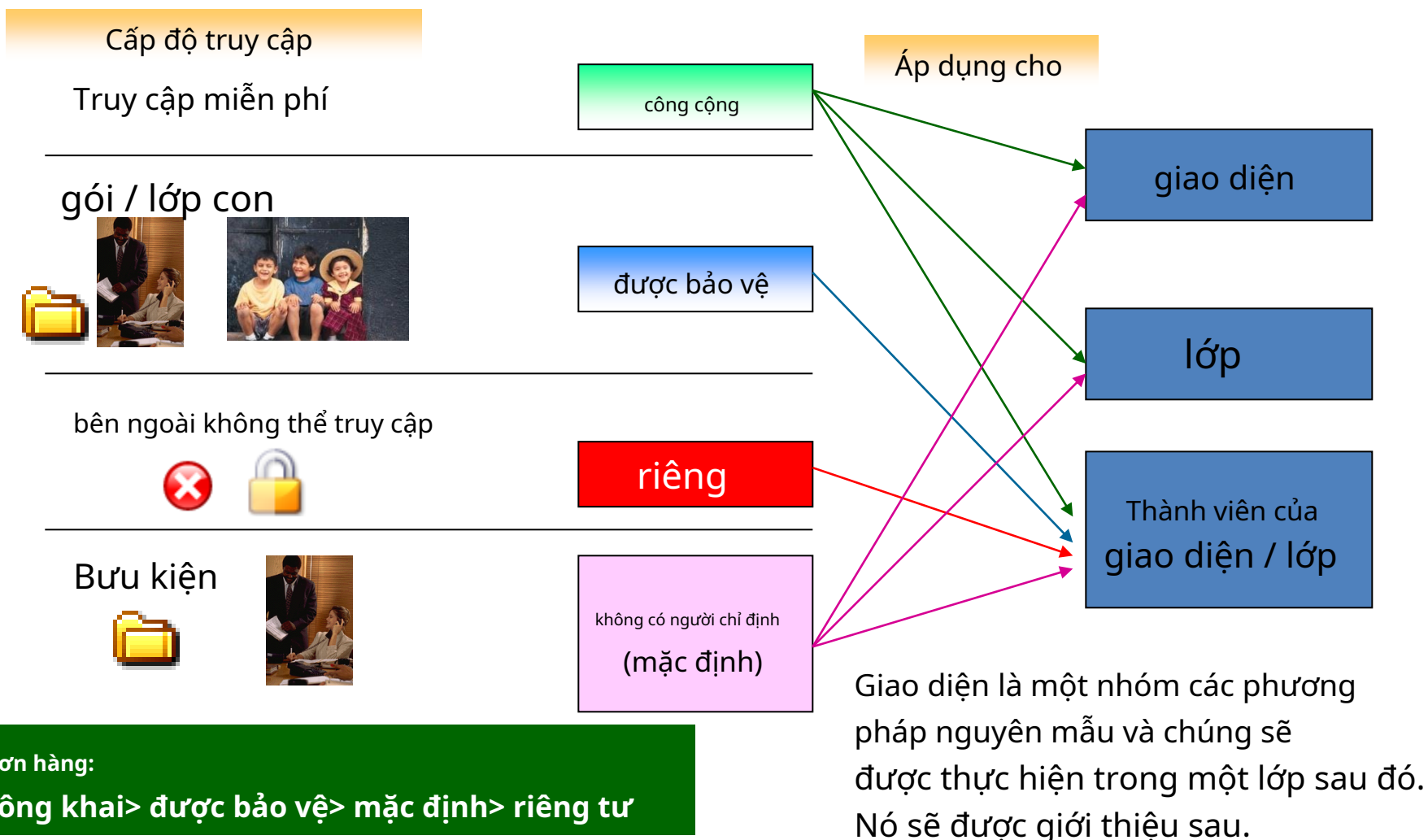
- Sợi dây **toString ()** Phương thức này trả về một biểu diễn chuỗi của đối tượng.
- boolean **bằng**(Đối tượng) Phương thức này cho biết liệu một số đối tượng khác có "bằng" đối tượng này hay không.
- int **Mã Băm()** Phương thức này trả về một giá trị mã băm cho đối tượng.
- Lớp **getClass ()** Phương thức này trả về lớp thời gian chạy của Đối tượng này.



Các bổ ngữ phổ biến - 1

- Bổ ngữ (ngôn ngữ học) là một từ có thể mang lại ý nghĩa của từ khác (tính từ → danh từ, trạng từ → động từ)
- Modifier (OOP) là từ khóa cung cấp cho trình biên dịch thông tin về bản chất của mã (phương thức), dữ liệu, lớp.
- Java hỗ trợ một số công cụ sửa đổi, trong đó một số công cụ trong số chúng là phổ biến và chúng được gọi là truy cập modifier (công khai, được bảo vệ, mặc định, riêng tư).
- Các công cụ sửa đổi thông thường sẽ áp đặt cấp độ truy cập vào
 - lớp (nó có thể được sử dụng ở đâu?)
 - các phương thức (cho dù chúng có thể được gọi hay không)
 - các trường (cho dù chúng có thể được đọc / ghi hay không)

Các bộ ngữ phổ biến - 2



Cấp độ truy cập

bổ nghĩa	Lớp	Tương tự Bưư kiện	Lớp con- Ngoài Bưư kiện	Thế giới
riêng	Y	n	n	n
Không (mặc định)	Y	Y	n	n
được bảo vệ	Y	Y	Y	n
công cộng	Y	Y	Y	Y

Mẹo chọn cấp độ truy cập

- Sử dụng **hạn chế nhất** cấp độ truy cập tạo ra **ý nghĩa đối với một thành viên cụ thể**. Sử dụng chế độ riêng tư trừ khi bạn có lý do chính đáng để không làm như vậy.
- **Tránh các trường công khai** ngoại trừ các hằng số. Các trường công khai có xu hướng liên kết bạn với một triển khai cụ thể và hạn chế tính linh hoạt trong việc thay đổi mã của bạn.

Phương pháp quá tải

```
/* Overloading methods Demo. */
```

```
public class Box {
```

```
    int length=0;
```

```
    int width=0;
```

```
    int depth=0;
```

```
    // Overloading constructors
```

```
    public Box() {
```

```
    }
```

```
    public Box(int l) {
```

```
        length = l>0? l: 0; // safe state
```

```
    }
```

```
    public Box(int l, int w) {
```

```
        length = l>0? l: 0; // safe state
```

```
        width = w>0? w: 0;
```

```
    }
```

```
    public Box(int l, int w, int d) {
```

```
        length = l>0? l: 0; // safe state
```

```
        width = w>0? w: 0;
```

```
        depth = d>0? d: 0;
```

```
    }
```

```
    // Overloading methods
```

```
    public void setEdge (int l,int w) {
```

```
        length = l>0? l: 0; // safe state
```

```
        width = w>0? w: 0;
```

```
    }
```

```
    public void setEdge (int l,int w,int d) {
```

```
        length = l>0? l: 0; // safe state
```

```
        width = w>0? w: 0;
```

```
        depth = d>0? d: 0;
```

```
    }
```

```
    public void output() {
```

```
        String S= "[" + length + "," + width
```

```
                + "," + depth + "];
```

```
        System.out.println(S);
```

```
    }
```

```
    /* Use the class Box */
```

```
    public class BoxUse {
```

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

```
            Box b= new Box();
```

```
            b.output();
```

```
            b.setEdge(7,3);
```

```
            b.output();
```

```
            b.setEdge(90,100,75);
```

```
            b.output();
```

```
        }
```

```
    }
```

```
    // classes và các đối tượng
```

Output - FirstPrj (run) x



run:

[0,0,0]

[7,3,0]

[90,100,75]

Access Modifier bị ghi đè

Projects: Chapter02

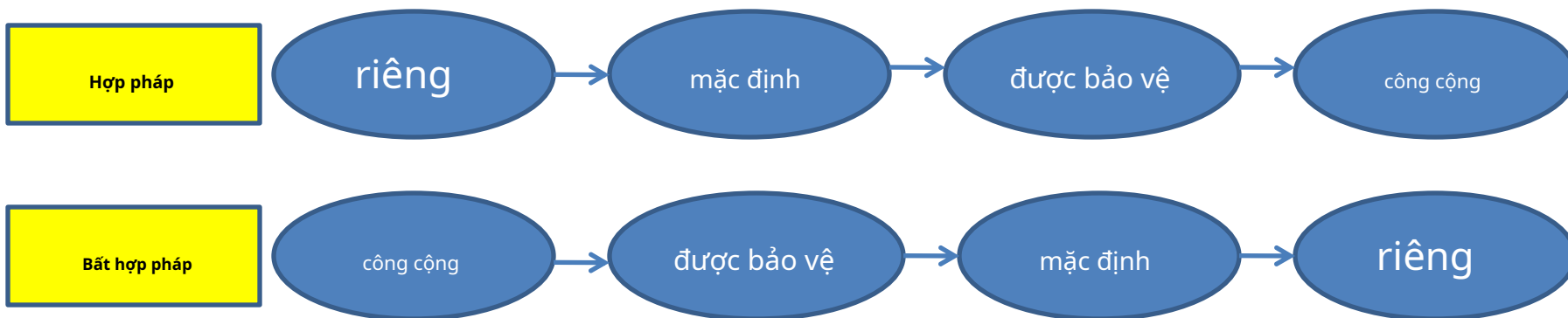
- Source Packages
 - boxPkg
 - Box.java
 - overridenDemo
 - ClassA.java
 - ClassB.java
 - rectPkg
 - Rectangle.java

ClassA.java

```
1 package overridenDemo;
2 public class ClassA {
3     public void M1() { }
4     protected void M2() { }
5     void M3() { }
6     private void M4() { }
7 }
8
```

ClassB.java

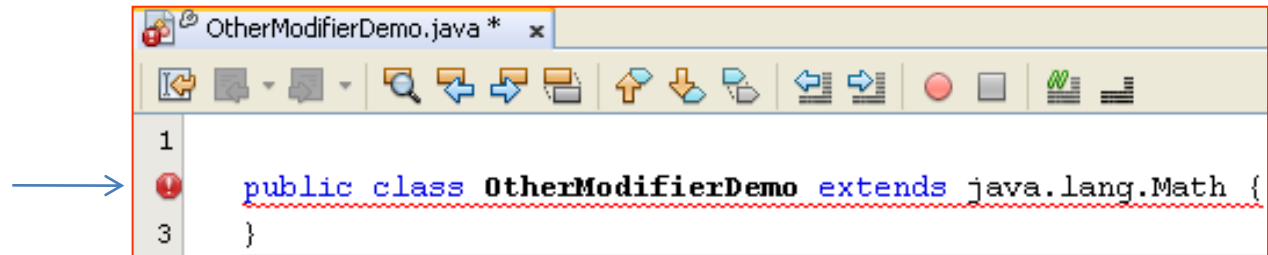
```
1 package overridenDemo;
2 public class ClassB extends ClassA {
3     protected void M1() { }
4     void M2() { }
5     private void M3() { }
6     void M4() { }
7 }
```



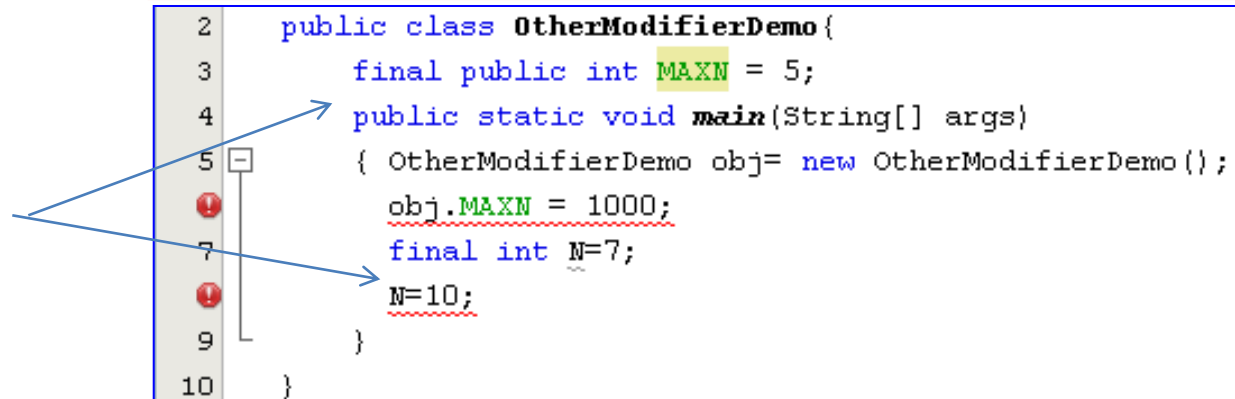
Lớp con phải được mở nhiều hơn lớp cha của nó

bổ nghĩa cuối cùng

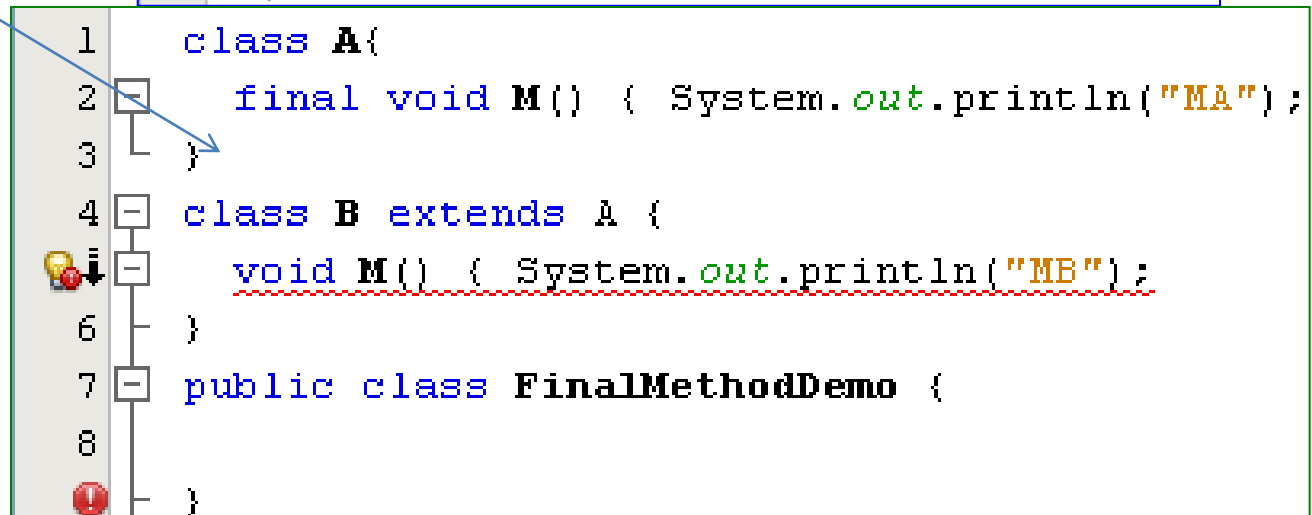
- **Lớp cuối cùng:**
Lớp không thể có phụ lớp
- **Dữ liệu cuối cùng** là một hằng số.
- **Phương pháp cuối cùng:**
một phương pháp có thể không được ghi đè.



```
1 public class OtherModifierDemo extends java.lang.Math {
3 }
```



```
2 public class OtherModifierDemo{
3     final public int MAXN = 5;
4     public static void main(String[] args)
5     { OtherModifierDemo obj= new OtherModifierDemo();
6       obj.MAXN = 1000;
7       final int N=7;
8       N=10;
9     }
10 }
```



```
1 class A{
2     final void M() { System.out.println("MA");
3 }
4 class B extends A {
5     void M() { System.out.println("MB");
6 }
7 public class FinalMethodDemo {
8 }
9 }
```


bổ nghĩa *tĩnh*

Biến lớp / Biến đối tượng

- Biến đối tượng: Biến của từng đối tượng
- Biến lớp: Một biến được chia sẻ trong tất cả các đối tượng của lớp. Nó được lưu trữ riêng biệt. Nó được khai báo với công cụ sửa đổi ***tĩnh***
- Biến lớp được lưu trữ riêng biệt. Vì vậy, nó có thể được truy cập như:

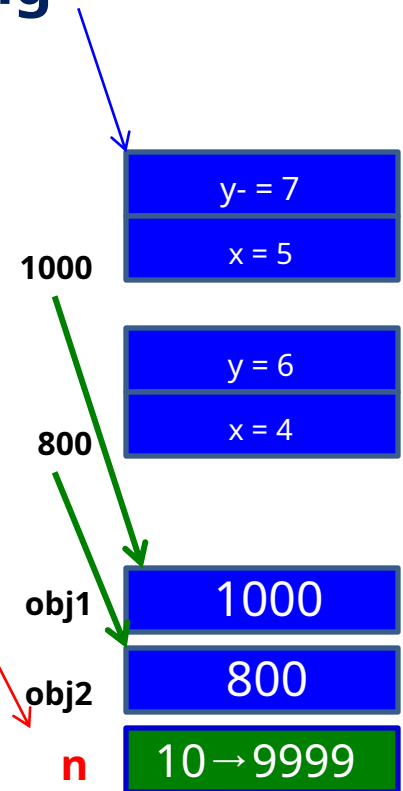
`object.staticVar`

`ClassName.staticVar`

bổ nghĩa *tĩnh*: Biến lớp / Biến đối tượng

```
public class StaticVarDemo {  
    static int N=10; // class variable  
    int x, y; // object variable  
    public StaticVarDemo(int xx, int yy){  
        x= xx; y=yy;  
    }  
    public void setN( int nn){  
        N= nn;  
    }  
    public void output(){  
        System.out.println(N + "," + x + "," + y);  
    }  
}
```

```
public class StaticVarDemoUse {  
    public static void main(String args[]){  
        StaticVarDemo obj1= new StaticVarDemo(5,7);  
        StaticVarDemo obj2= new StaticVarDemo(4,6);  
        obj1.output();  
        obj2.output();  
        obj1.setN(9999);  
        obj1.output();  
        obj2.output();  
        System.out.println(StaticVarDemo.N);  
    }  
}
```



Output - FirstPrj (run) x

```
run:  
10,5,7  
10,4,6  
9999,5,7  
9999,4,6  
9999
```

bổ nghĩa *tĩnh*: mã tĩnh - Khối nổi miễn phí

```
public class StaticCodeDemo {  
    public static int N=10;  
    int x=5, y=7;  
    static {  
        System.out.println("Static code:" + N);  
    }  
    int sum(){  
        return x+y;  
    }  
    static {  
        System.out.println("Static code: Hello");  
    }  
}
```

```
public class StaticCodeDemoUse {  
    public static void main(String args[]){  
        System.out.println(StaticCodeDemo.N);  
        StaticCodeDemo obj= new StaticCodeDemo();  
        System.out.println(obj.sum());  
    }  
}
```

Tất cả mã tĩnh chỉ chạy
một lần khi mã đầu tiên

thời gian của lớp học
chứa chúng là
truy cập

Lần truy cập thứ hai

Output - FirstPrj (run) x

run:
Static code:10
Static code: Hello
10
12

bổ nghĩa *tĩnh*: Phương pháp tĩnh

- Nó được gọi là phương thức lớp / phương thức toàn cục và nó được gọi mặc dù không có đối tượng nào của lớp này được tạo.
- Điểm đầu vào của chương trình Java là một phương thức tĩnh
- Cú pháp để gọi nó: **ClassName.staticMethod (args)**
- ***Phương pháp tĩnh:***
 - có thể truy cập trực tiếp các biến lớp và phương thức lớp **chỉ một**.
 - ***không thể*** truy cập trực tiếp các biến cá thể hoặc phương thức cá thể — chúng phải sử dụng tham chiếu đối tượng.
 - ***không thể*** sử dụng từ khóa this vì không có trường hợp nào cho từ khóa này để tham khảo

Những gì nên được tính trong một lớp?

- Hằng số:
 - Công cụ sửa đổi tĩnh, kết hợp với công cụ sửa đổi cuối cùng, cũng được sử dụng để định nghĩa các hằng số. Công cụ sửa đổi cuối cùng chỉ ra rằng giá trị của trường này không thể thay đổi.

*PI kép cuối cùng tĩnh =
3,141592653589793;*

Phương pháp với Số lượng đối số tùy ý

Một nhóm được coi là một mảng
group.length → số phần tử **nhóm [i]**:
Phần tử ở vị trí i

```
1 public class ArbitraryDemo {
2     public double sum(double... group){
3         double S=0;
4         for (double x: group) S+=x;
5         return S;
6     }
7     public String concatenate(String... group){
8         String S="";
9         for (String x: group) S+=x + " ";
10        return S;
11    }
12    public static void main(String[] args){
13        ArbitraryDemo obj= new ArbitraryDemo();
14        double total= obj.sum(5.4, 3.2, 9.08, 4);
15        System.out.println(total);
16        String line = obj.concatenate("I", "love", "you", "!");
17        System.out.println(line);
18    }
19 }
```

Output - FirstPrj (run) x

run:
21.68
I love you !

Example: một phương tiện

giai cấp công cộng **SingleVehicle**{

phương tiện công cộng **đầu vào()** {

Phương tiện v;

Nhà sản xuất dây;

int năm;

chi phí gấp đôi;

Màu chuỗi;

Máy quét trong = Máy quét mới (System.in);

System.out.print ("Nhà sản xuất:"); nhà sản
xuất = in.nextLine ();

System.out.print ("Năm sản xuất:");

```
năm = Integer.parseInt (in.nextLine ());  
System.out.print ("Chi phí:");  
cost = Double.parseDouble (in.nextLine ());  
System.out.print ("Màu:");  
color = in.nextLine ();  
v = Xe mới (nhà sản xuất, năm, chi phí,  
    màu sắc);  
trả lại v;  
    }  
khoảng trống công cộng đầu ra(Xe v) {  
    System.out.println (v.toString ());  
}}
```


Ví dụ: Mảng phương tiện

giai cấp công cộng **ArrayVehicle**{

 phương tiện công cộng [] **đầu vào**(int n) {

 Xe [] v = new Xe [n]; Nhà sản xuất
 dây;

 int năm;

 chi phí gấp đôi;

 Màu chuỗi;

 Máy quét trong = Máy quét mới (System.in);

 for (int i = 0; i < n; i++) {

 System.out.print ("Nhà sản xuất:"); nhà sản
 xuất = in.nextLine ();

```

System.out.print ("Năm sản xuất:"); năm =
Integer.parseInt (in.nextLine ()); System.out.print
("Chi phí:");
cost = Double.parseDouble (in.nextLine ());
System.out.print ("Màu:");
color = in.nextLine ();
v [i] = Xe mới (nhà sản xuất, năm, chi phí,
    màu sắc);}
    trả lại v;    }
    khoảng trống công cộng đầu ra(Xe [] v) {
        System.out.println ("Màu năm của nhà sản xuất");
        for (int i = 0; i <v.length; i ++)
            System.out.println (v [i] .toString ());
    }}

```

Trị giá

Sử dụng đơn giản các lớp bên trong

- **Các lớp bên trong** là các lớp được định nghĩa trong các lớp khác
 - Lớp bao gồm lớp bên trong được gọi là **lớp ngoài**
 - Không có vị trí cụ thể nào mà định nghĩa của lớp bên trong (hoặc các lớp) phải được đặt trong lớp bên ngoài
 - Đặt nó trước hoặc cuối cùng, tuy nhiên, sẽ đảm bảo rằng nó dễ dàng tìm thấy

Sử dụng đơn giản các lớp bên trong

- Định nghĩa lớp bên trong là thành viên của lớp bên ngoài giống như cách mà các biến thể hiện và phương thức của lớp bên ngoài là thành viên
 - Lớp bên trong là cục bộ đối với định nghĩa lớp bên ngoài
 - Tên của lớp bên trong có thể được sử dụng lại cho một thứ khác bên ngoài định nghĩa lớp bên ngoài
 - Nếu lớp bên trong là private, thì lớp bên trong không thể được truy cập bằng tên bên ngoài định nghĩa của lớp bên ngoài

Lớp bên trong / bên ngoài

giai cấp công cộng

{

lớp riêng bên trong

{

// biến cá thể lớp bên trong

// phương thức lớp bên trong

} // kết thúc định nghĩa lớp bên trong

// biến cá thể lớp ngoài // các phương thức

lớp ngoài

}

Sử dụng đơn giản các lớp bên trong

- Có hai ưu điểm chính đối với các lớp bên trong
 - Chúng có thể làm cho lớp bên ngoài trở nên khép kín hơn vì chúng được định nghĩa bên trong một lớp
 - Cả hai phương thức của chúng đều có quyền truy cập vào các phương thức riêng và các biến cá thể của nhau
- Sử dụng lớp bên trong làm lớp trợ giúp là một trong những ứng dụng hữu ích nhất của lớp bên trong
 - Nếu được sử dụng như một lớp trợ giúp, một lớp bên trong phải được đánh dấu là riêng tư

Đồ bóng (1)

- Khai báo của một loại trong một phạm vi cụ thể có cùng tên với một khai báo khác trong phạm vi kèm theo, sau đó từ khai ***bóng tối*** khai báo phạm vi bao quanh.

Biến đồ bóng (2)

```
public class ShadowTest {  
    public int x = 0;  
    class FirstLevel {  
        public int x = 1; // shadowing  
  
        void methodInFirstLevel(int x) {  
            System.out.println("x = " + x);  
            System.out.println("this.x = " + this.x);  
            System.out.println("ShadowTest.this.x = " + ShadowTest.this.x);  
        }  
    }  
  
    public static void main(String... args) {  
        ShadowTest st = new ShadowTest();  
        ShadowTest.FirstLevel fl = st.new FirstLevel();  
        fl.methodInFirstLevel(23);  
    }  
}
```

Output - FirstPrj (run) x

run:
x = 23
this.x = 1
ShadowTest.this.x = 0

Cơ chế: Dữ liệu cục bộ được xử lý trước tiên

Thực thi

Mối quan hệ hướng đối tượng

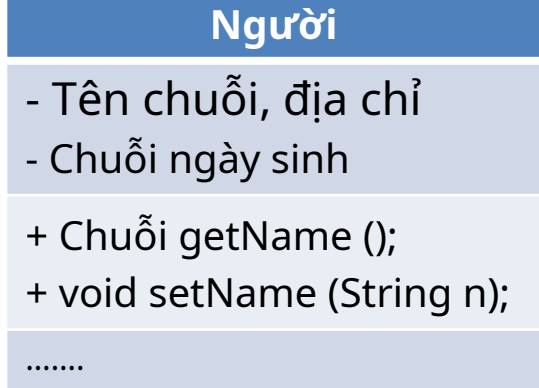
- 3 quan hệ chung trong các lớp:
 - "là một / một loại"
 - "có một"
 - sự kết hợp
- Ví dụ:
 - Học sinh **là một** người
 - "Một ngôi nhà là một ngôi nhà mà **có một** gia đình và một con vật cưng."
 - Một Giáo sư có một số sinh viên và một giáo sư có thể được chứa trong một Sinh viên

Thực thi

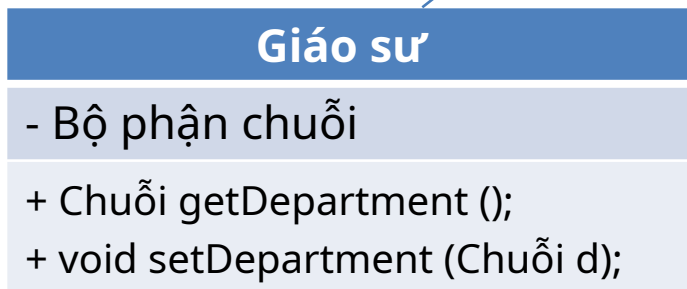
Mỗi quan hệ hướng đối tượng...

Quan hệ “là a” được triển khai dưới dạng hạng phụ

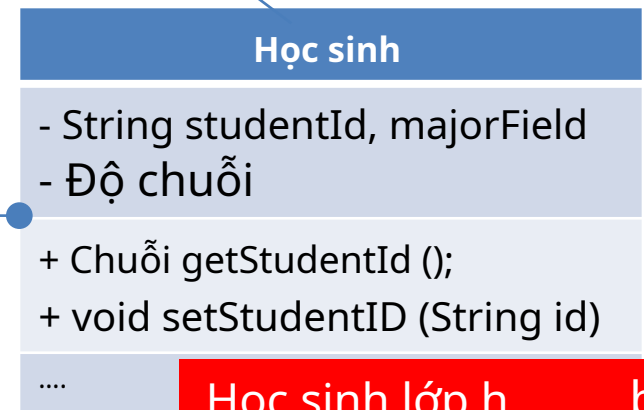
Giáo sư lớp học, Sinh viên là các lớp phụ của lớp Person Lớp con kế thừa cấu trúc của siêu lớp



Quan hệ “có a” được triển khai dưới dạng
thẩm quyền giải quyết



Lớp Giáo sư có trường Sinh viên [] sinh viên



Học sinh lớp h bắt
lĩnh vực giáo sư pr

dạy

là một

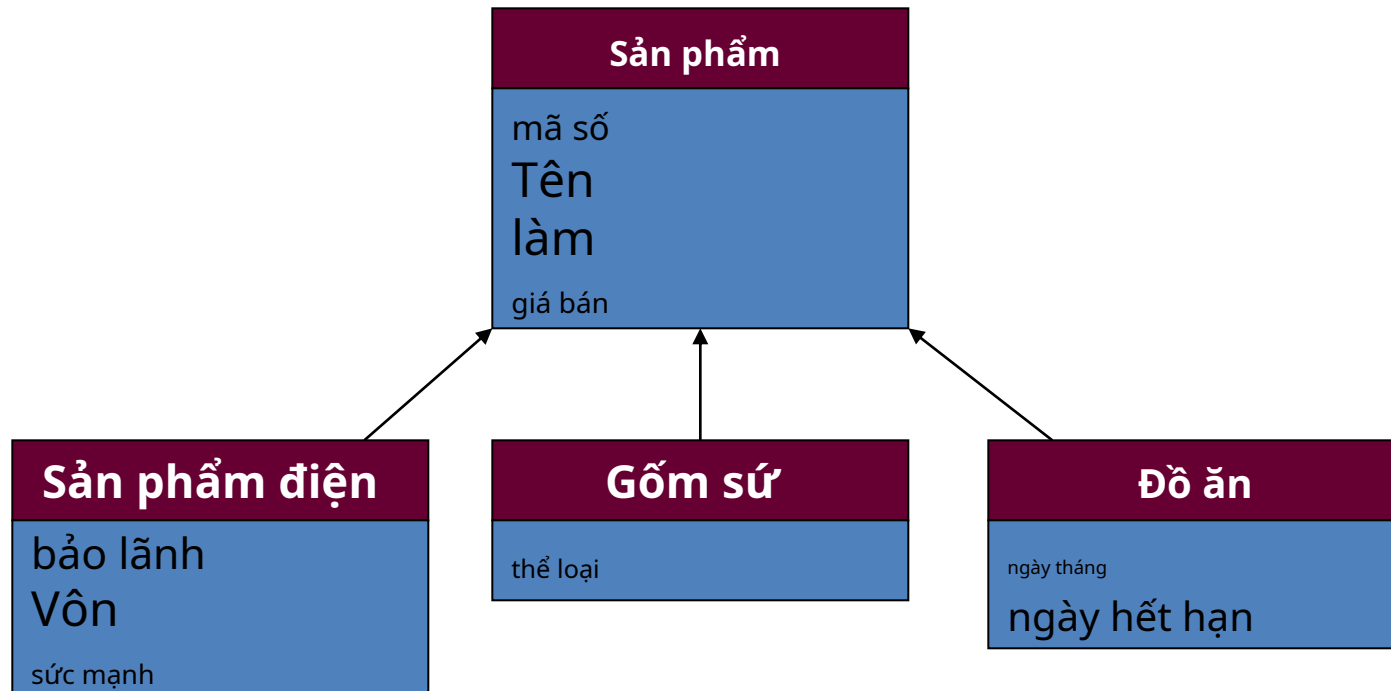
Di sản

- Có một số lớp con từ **một** siêu cấp → Kế thừa là một mối quan hệ mà các đối tượng **chia sẻ một cấu trúc chung**: cấu trúc của một đối tượng này là cấu trúc con của một đối tượng khác.
- Các **kéo dài** từ khóa được sử dụng để tạo lớp con.
- Một lớp có thể được dẫn xuất trực tiếp từ **chỉ một** lớp (Java là một **thừa kế duy nhất** Ngôn ngữ OOP).
- Nếu một lớp không có bất kỳ siêu lớp nào, thì nó được dẫn xuất ngầm từ lớp Đối tượng.
- Không giống như các thành viên khác, hàm tạo **không thể được kế thừa** (hàm tạo của lớp siêu không thể khởi tạo các đối tượng của lớp con)

Di sản...

- **Làm thế nào để xây dựng một cấu trúc phân cấp lớp? → Ngã tư**

- Sản phẩm điện <mã, tên, sản xuất, giá cả, đảm bảo, điện áp, điện>
- Sản phẩm gốm <mã, tên, sản xuất, giá cả, gỗ>
- Sản phẩm thực phẩm <mã, tên, sản xuất, giá cả, ngày, hết hạn>



Kế thừa...: Từ khóa “siêu”

- Người xây dựng là **Không** Thừa hưởng
- super (...) để tái sử dụng khối mã lệnh
 - siêu (đối số); *//gọi một hàm tạo siêu lớp*
 - Cuộc gọi ***cần phải là đầu tiên*** câu lệnh trong **hàm tạo lớp con**
- Thay thế hàm tạo không có tham số mặc định

Kế thừa...: Từ khóa “siêu”

- Chúng tôi sử dụng từ khóa `super` trong Java làm định nghĩa cho một lệnh gọi phương thức:
siêu. methodName (các đối số);
- Bất cứ khi nào chúng ta muốn gọi phiên bản của `methodName` đã được định nghĩa bởi lớp cha của chúng ta.
- **siêu()** được sử dụng để truy cập phương thức khởi tạo của lớp cha. Và Nó phải là câu lệnh đầu tiên trong phương thức khởi tạo của lớp con.

Di sản...

```
1 public class Rectangle {
2     private int length = 0;
3     private int width = 0;
4     // Overloading constructors
5     public Rectangle() // Default constructor
6     {
7     }
7 public Rectangle(int l, int w)
8     {
8     length = l>0? l: 0;    width= w>0? w: 0;
9     }
10    // Overriding the toString method of the java.lang.Object class
11    public String toString()
12    {
12    return "[" + getLength() + "," + getWidth() + "]\n";
13    }
14    // Getters, Setters
15    public int getLength() { return length; }
16    public void setLength(int length) { this.length = length; }
17    public int getWidth() { return width; }
18    public void setWidth(int width) { this.width = width; }
19    public int area() { return length*width; }
20 }
```

Di sản...

```
1 public class Box extends Rectangle {
2     private int height=0; // additional data
3     public Box() { super(); }
4     public Box (int l, int w, int h)
5     { super(l, w); // Try swapping these statements
6       height = h>0? h: 0;
7     }
8     // Additional Getter, Setter
9     public int getHeight() { return height; }
10    public void setHeight(int height)
11    { this.height = height; }
12    // Overriding methods
13    public String toString()
14    { return "[" + getLength() + "," +
15      getWidth() + "," + getHeight() + "]";
16    }
17    public int area() {
18        int l = this.getLength();
19        int w = this.getWidth();
20        int h = this.getHeight();
21        return 2*(l*w + w*h + h*l);
22    }
23    // additional method
24    public int volumn() {
25        return this.getLength()*this.getWidth()*height;
26    }
27 }
```

```
1 public class Demo_1 {
2     public static void main (String[] args)
3     { Rectangle r= new Rectangle(2,5);
4       System.out.println("Rectangle: " + r.toString());
5       System.out.println("  Area: " + r.area());
6       Box b= new Box(2,2,2);
7       System.out.println("Box " + b.toString());
8       System.out.println("  Area: " + b.area());
9       System.out.println("  Volumn: " + b.volumn());
10    }
11 }
```

Output - Chapter06 (run)

```
run:
Rectangle: [2,5]
Area: 10
Box [2,2,2]
Area: 24
Volumn: 8
BUILD SUCCESSFUL (total time: 0 seconds)
```


lớp công cộng Điểm {

int riêng x, y;

public Point (int x, int y) {

 this.x = x; this.y = y;}

... ..// getter & setter} Đa giác

lớp công khai {

được bảo vệĐiểmd1, d2, d3, d4; public void setD1

(Point d1) {this.d1 = d1;} public Point getD1 () {return
d1;}

@Ghi đề

công cộngSợi dâytoString (){

 trở về

d1.getX () + "\ t" + d1.getY () + "\ t" + d2.getX () + "\ t" + d2.ge tY () +

"\ t" + d3.getX () + "\ t" + d3.getY () + "\ t" + d4.getX () + "\ t" + d4.getY
(); }}

giai cấp công cộng Quảng trường mở rộng **Đa giác**{

Quảng trường công cộng(){

d1=new Point (0,0);**d2**=new Point (0,1); **d3**=Điểm mới
(1,0);**d4**=new Point (1,1);

}

}

lớp công khai Chính {

public static void main (String args []) {

Square sq = new Square (); System.out.println
(sq.**toString ()**);

}

}

```
hạng công khai Người {  
    riêngSợi dây      Tên;  
    riêngSợi dây      thứ hai;  
    Public Person ()    {  
    } // getter & setter} public class Nhân viên mở
```

```
rộngNgười{  
    lương gấp đôi tư nhân;  
    public double getSalary () {trả lại tiền lương; } public void  
    setSalary (lương gấp đôi) {  
        this.salary = tiền lương; } @Ghi đề
```

```
    public String toString () {  
        //trở vềTên+ "\t "+ngày sinh nhật+ "\t "+ tiền lương;  
        trở vềsuper.getName ()+ "\t "+  
        super.getBithday ()+ "\t "+ tiền lương;  
    }}
```

```
lớp công khai Chính {  
    public static void main (String args []) {  
  
        Nhân viên e = Nhân viên mới (); e.đặt  
        tên("Tô Ngọc Vân");  
        e.setBithday("3/4/1994");  
        e.setSalary(4,4);  
        System.out.println (e.);  
    }  
}
```

Đầu ra ???

1.giai cấp công cộng **Đa giác**{ được bảo vệ
Điểm d1, d2, d3, d4; công cộng **Đa giác** (){

System.out.println ("Lớp đa giác");}}

2.giai cấp công cộng **Quảng trường** kéo dài **Đa giác**{ công
cộng **Quảng trường**() {
System.out.println ("Lớp vuông");}}

3.lớp công khai Chính {
public static void main (String args []) {
Square hv = new Square ();
}}

1.giai cấp công cộng **Đa giác**{ được bảo vệ
Điểm d1, d2, d3, d4;
// một hàm tạo không phải là hàm tạo mặc định}

2.giai cấp công cộng **Quảng trường** kéo dài **Đa giác**{ công
cộng **Quảng trường**() {
System.out.println ("Lớp vuông");}}

3.lớp công khai Chính {
public static void main (String args []) {
Square hv = new Square ();
}}

Tại sao Lỗi ???

Phương pháp ghi đè và ẩn (1)

- **Ghi đè một phương thức:** Một phương thức thể hiện trong lớp con có cùng chữ ký (tên, cộng với số và kiểu tham số của nó) và kiểu trả về như một phương thức thể hiện trong lớp cha sẽ ghi đè phương thức của lớp cha.
 - Sử dụng **@Ghi đè** chú thích hướng dẫn trình biên dịch mà bạn định ghi đè một phương thức trong lớp cha (bạn có thể không sử dụng nó vì ghi đè là mặc định trong Java).
- **Ẩn một phương pháp:** Triển khai lại một phương thức tĩnh được triển khai trong siêu lớp

Phương pháp (2)

```
class Father1 {  
    public static void m(){  
        System.out.println("I am a father");  
    }  
}  
  
class Son1 extends Father1{  
    public static void m(){  
        System.out.println("I am a son");  
    }  
}  
  
public class HidingMethodDemo {  
    public static void main(String args[]){  
        Father1 obj= new Father1();  
        obj.m();  
        obj= new Son1();  
        obj.m();  
        Son1 obj2= new Son1();  
        obj2.m();  
    }  
}
```

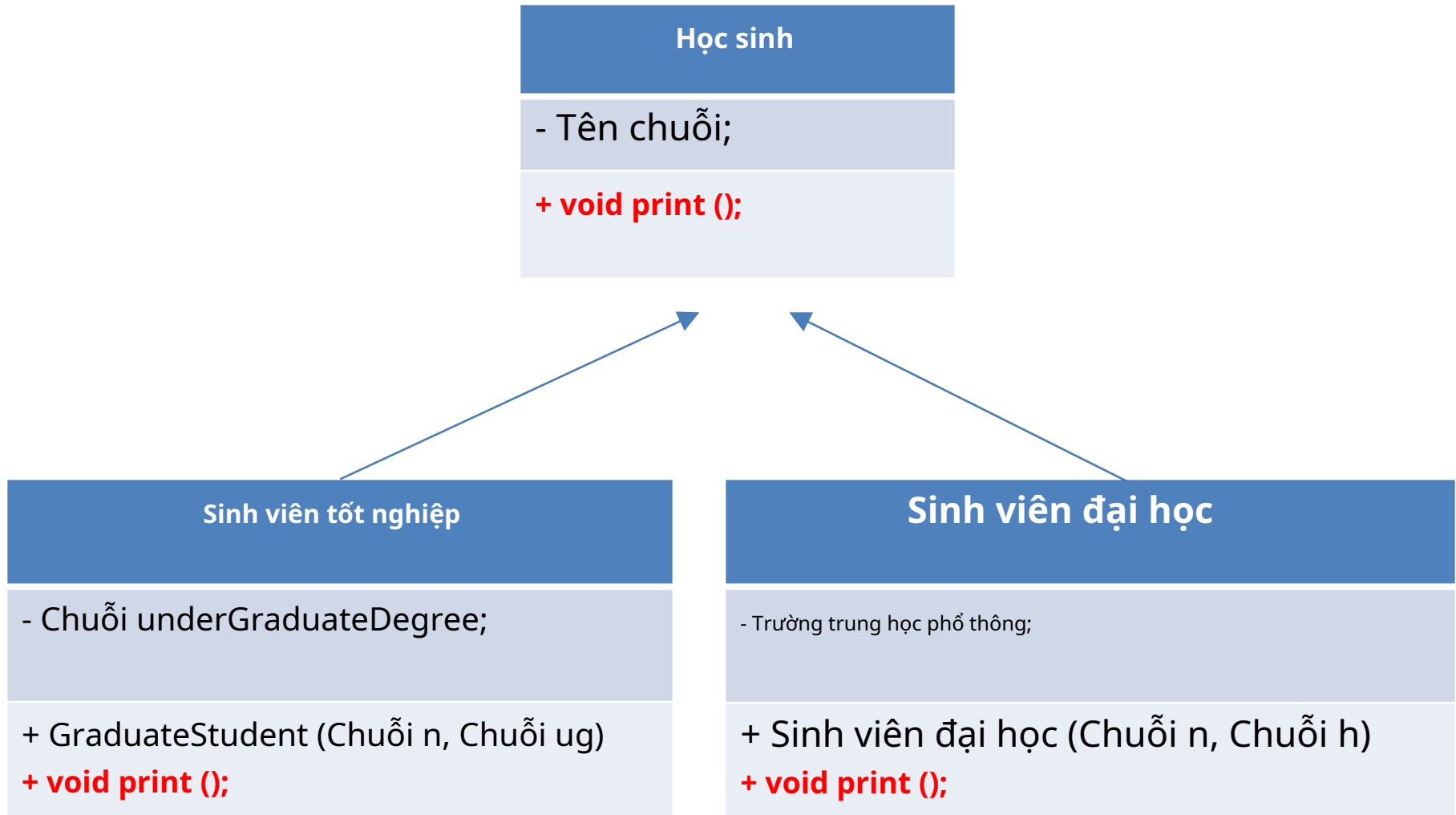
Output - FirstPrj (run) x

run:
I am a father
I am a father
I am a son

Tính đa hình

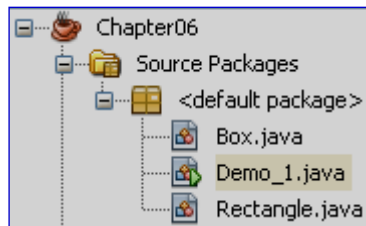
- Khả năng của hai hoặc nhiều đối tượng thuộc **khác nhau** các lớp học để phản hồi chính xác **tương tự** tin nhắn (cuộc gọi phương thức) theo những cách khác nhau dành riêng cho từng lớp.
- ***Kế thừa kết hợp với ghi đè tạo điều kiện cho tính đa hình.***

Đa hình...



Ghi đè các phương thức kế thừa

Phương thức ghi đè: Một phương thức kế thừa được viết lại



```
1 public class Rectangle {
2     protected int length=0, width=0;
3     // Overloading methods
4     public void setValue(int l)
5     { length = l>0?l:0; }
6     public void setValue(int l, int w)
7     { length = l>0? l: 0;
8       width= w>0? w: 0; }
9 }
10 // Overriding the toString method of the java.lang.Object class
11 public String toString()
12 { return "[" + length + "," + width + "]"; }
13 }
14 }
15 }
```

```
1 public class Box extends Rectangle {
2     int height=0;
3     public void set (int l, int w, int h)
4     { super.setValue(l, w);
5       height = h>0? h: 0;
6     }
7     // Overriding the toString method
8     // of the Rectangle class
9     public String toString()
10    { return "[" + length + "," + width +
11      "," + height + "]"; }
12 }
13 }
```

Output - Chapter06 (run)

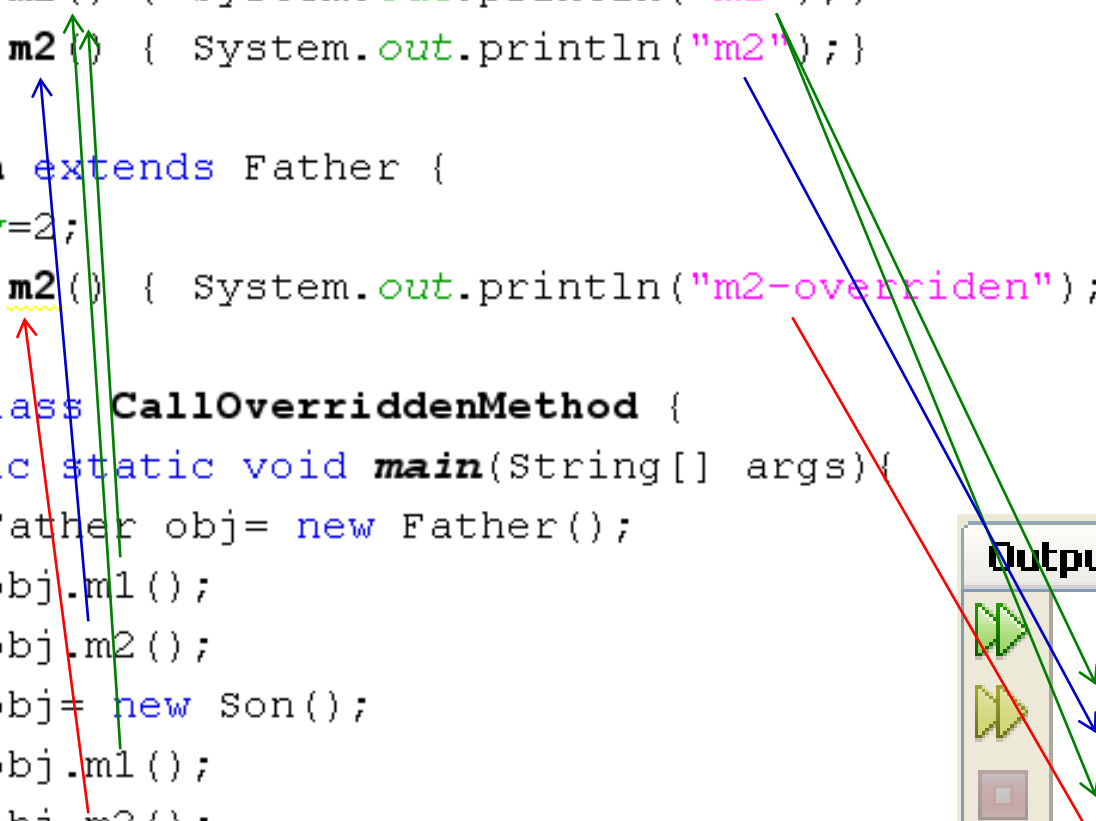
```
run:
[5,0]
[10,20]
[5,10,15]
```

```
public class Demo_1 {
    public static void main (String[] args)
    { Rectangle r= new Rectangle();
      r.setValue(5);
      System.out.println(r.toString());
      r.setValue(10,20);
      System.out.println(r.toString());
      Box b= new Box();
      b.set(5,10,15);
      System.out.println(b.toString());
    }
}
```

Phương thức bị quá tải: Các phương thức có cùng tên nhưng các tham số của chúng khác nhau trong một lớp

Làm thế nào có thể được ghi đè phương pháp

```
class Father{
    int x=0;
    void m1() { System.out.println("m1");}
    void m2() { System.out.println("m2");}
}
class Son extends Father {
    int y=2;
    void m2() { System.out.println("m2-overridden");}
}
public class CallOverriddenMethod {
    public static void main(String[] args){
        Father obj= new Father();
        obj.m1();
        obj.m2();
        obj= new Son();
        obj.m1();
        obj.m2();
    }
}
```



```
Output - FirstPrj (run) x
run:
m1
m2
m1
m2-overridden
```

Example

- Mã một lớp có tên **Xe ô tô**, **Động cơ**, **Xe tải** được dẫn xuất trực tiếp từ lớp cơ sở **Phương tiện giao thông** vật.

giai cấp công cộng **Xe ô tô** kéo dài **Phương tiện giao thông** {

riêng String typeofEngine;

riêng ghế int;

công cộng **Xe ô tô**() {

siêu();

}

```

công cộng Xe ô tô(Nhà sản xuất chuỗi, int
    năm, chi phí gấp đôi, màu chuỗi, chuỗi
    typeofEngine, ghế int) {
    siêu(nhà sản xuất, năm, chi phí, màu sắc);
    this.typeofEngine = typeofEngine; this.seats = chỗ
    ngồi;
}

chuỗi công khai gettypeofEngine() {
    trả về typeofEngine;
}

khoảng trống công cộng settypeofEngine(Sợi dây
    typeofEngine) {
    this.typeofEngine = typeofEngine;
}

```

```
int công cộng getSeats() {  
    trả lại chỗ ngồi;  
}
```

```
khoảng trống công cộng setSeats(chỗ ngồi) {  
    this.seats = chỗ ngồi;  
}
```

```
chuỗi công khai() {  
    trở về  
    siêu.toString () + "\ t" + typeofEngine + "\ t" +  
    chỗ ngồi;  
}}
```

Ví dụ: Mảng ô tô, động cơ, xe tải

```
giai cấp công cộng ArrayVehicles{  
    danh sách [] xe riêng; int n  
    riêng;  
    công cộng ArrayVehicles() {  
        n = 0;  
        list = new Xe [50];  
    }  
    int công cộng getNumberOfVehicle() {  
        trả về n;  
    }  
}
```



```
phương tiện cá nhânđầu vào() {  
    Phương tiện v;  
    Nhà sản xuất dây;  
    int năm; chi phí gấp đôi; Màu  
    chuỗi;  
    Máy quét trong = Máy quét mới (System.in);  
    System.out.print ("Nhà sản xuất:"); nhà sản  
    xuất = in.nextLine ();  
    System.out.print ("Năm sản xuất:"); năm =  
    Integer.parseInt (in.nextLine ()); System.out.print  
    ("Chi phí:");  
    cost = Double.parseDouble (in.nextLine ());  
    System.out.print ("Màu:");  
    color = in.nextLine ();  
    v = Xe mới (nhà sản xuất, năm, chi phí, màu sắc);  
  
    trả lại v;  
}
```

```
khoảng trống công cộnginputMotor() {  
    năng lượng kép;  
    Xe v = input ();  
    Máy quét trong = Máy quét mới (System.in);  
    System.out.print ("Nguồn:");  
    power = in.nextDouble ();  
    list [n ++] = new  
    Động cơ (v.getManufacturer (), v.getYear (),  
    v.getCost (), v.getColor (), power);  
}
```

```

khoảng trống công cộnginputCar() {
    String typeofEngine;
    ghế int;
    Xe v = input ();
    Máy quét trong = Máy quét mới (System.in);
    System.out.print ("Loại động cơ:"); typeofEngine
    = in.nextLine ();
    System.out.print ("Số ghế:"); ghế = in.nextInt ();

    list [n ++] = new
    Xe hơi (v.getManosystem (), v.getYear (), v.getCost (),
    v.getColor (), typeofEngine, chỗ ngồi);

    }

```

```

int công cộng getNumberOfCar() {
    int count = 0;
    for (int i = 0; i < n; i++)
        if (list [i] instanceof Car)
            tính ++;
    số lượng trả lại; }

int công cộng getNumberOfMotor() {
    int count = 0;
    for (int i = 0; i < n; i++)
        if (list [i] instanceof Motor)
            tính ++;
    số lượng trả lại;
}

```

khoảng trống công cộng **đầu ra()** {

if (getNumberOfCar () > 0) {

System.out.println ("Danh sách của ô tô");

System.out.println ("Loại Màu Chi phí của Năm
Nhà sản xuất of Engine Number of Seats");

for (int i = 0; i < n; i++) {

if (list [i] instanceof Car) System.out.println (list
[i] .toString ());

}

System.out.println ("-----");

System.out.println ("Số lượng xe:" +
getNumberOfCar ());

}

```
if (getNumberOfMotor ()> 0) {  
    System.out.println ("Danh sách Động cơ");  
    System.out.println ("Công suất Màu Chi phí Năm  
của Nhà sản xuất");  
    for (int i = 0; i <n; i ++) {  
        if (list [i] instanceof Motor) System.out.println (list  
[i] .toString ());  
    }  
    System.out.println ("-----");  
    System.out.println ("Số lượng động  
cơ:" + getNumberOfMotor ());  
}
```

```
if (getNumberOfTruck ()> 0) {  
    System.out.println ("Danh sách Xe tải");  
    System.out.println ("Tải màu chi phí theo năm của  
nhà sản xuất");  
    for (int i = 0; i <n; i ++ ) {  
        if (list [i] instanceof Truck) System.out.println (list  
[i] .toString ());  
        }  
    System.out.println ("-----");  
    System.out.println ("Số lượng xe tải:" +  
getNumberOfTruck ());  
    }  
}
```

Thực đơn

```
lớp công khai Chính {  
    public static void main (String [] args) {
```

```
        ArrayVehicles a = new  
        ArrayVehicles ();
```

```
        Máy quét trong = mới  
        Máy quét (System.in);
```


trong khi (đúng) {

```
System.out.print ("\ n 1. nhập Xe"); System.out.print  
("\ n 2. đầu vào một Động cơ"); System.out.print ("\ n  
3. nhập Xe tải"); System.out.print ("\ n 4. đầu ra");  
System.out.print ("\ n 0. Thoát");
```

```
System.out.print ("\ n Lựa chọn của bạn (0-> 4):");
```

```
int lựa chọn;
```

```
lựa chọn = in.nextInt ();
```

```
chuyển đổi (lựa chọn) {
```

```
    trường hợp 1: a.inputCar ();
```

```
        nghỉ;
```

trường hợp 2: a.inputMotor ();

nghỉ;

trường hợp 3: a.inputTruck ();

nghỉ;

trường hợp 4: a.output ();

nghỉ;

trường hợp 0:

System.out.println ("Tạm biệt !!!!");

System.exit (0);

nghỉ;

default: System.out.println ("chỉ để chọn (0->4)");

}}}}

Giao diện

- Một *giao diện* là một kiểu tham chiếu, tương tự như một lớp, có thể chứa *chỉ một hằng số*, trường khởi tạo, phương thức tĩnh, nguyên mẫu (phương thức trừu tượng, phương thức mặc định), phương thức tĩnh và kiểu lồng nhau.
- Nó sẽ là **cốt lõi** của một số lớp học
- Không thể khởi tạo các giao diện vì chúng có **không ai** các phương pháp.
- Các giao diện chỉ có thể là *thực hiện* bởi các lớp học hoặc *mở rộng* bằng các giao diện khác.

Các giao diện...

```
1 public interface InterfaceDemo {
2     final int MAXN=100; // constant
3     int n=0; // Fields in interface must be initialized
4     static public int sqr(int x){ return x*x;}
5     public abstract void m1(); // abstract methods
6     abstract public void m2();
7     void m3(); // default methods
8     void m4();
9 }
10
11 class UseIt{
12     public static void main(String args[]){
13         InterfaceDemo obj= new InterfaceDemo();
14     }
15 }
```

Các giao diện...

```
public interface InterfaceDemo {  
    final int MAXN=100; // constant  
    int n=0; // Fields in interface must be initialized  
    static public int sqr(int x){ return x*x;}  
    public abstract void m1(); // abstract methods  
    abstract public void m2();  
    void m3(); // default methods  
    void m4();  
}  
  
class A implements InterfaceDemo{  
    // overriding methods  
    public void m1() { System.out.println("M1");}  
    public void m2() { System.out.println("M2");}  
    void m3() { System.out.println("M3");}  
    void m4() { System.out.println("M4");}  
}
```

m3 (), m4 () trong A
không thể thực hiện
m3 (), m4 () in
InterfaceDemo,
cố gắng chỉ định
truy cập yếu hơn
đặc quyền, đã
công cộng

Các phương thức mặc định của một giao diện phải được ghi đè như các phương thức công khai trong các lớp cụ thể.

```
public interface InterfaceDemo {
    final int MAXN=100; // constant
    int n=0; // Fields in interface must be initialized
    static public int sqr(int x){ return x*x;}
    public abstract void m1(); // abstract methods
    abstract public void m2();
    void m3(); // default methods
    void m4();
}
```

```
class A implements InterfaceDemo{
    // overriding methods
    public void m1() { System.out.println("M1");}
    public void m2() { System.out.println("M2");}
    public void m3() { System.out.println("M3");}
    public void m4() { System.out.println("M4");}
}
```

```
class UseIt{
    public static void main(String args[]){
        InterfaceDemo obj= new A();
        obj.m1();
        obj.m2();
        obj.m3();
        obj.m4();
        int s= InterfaceDemo.sqr(5);
        System.out.println("5x5=" + s);
    }
}
```

Output - FirstPrj (run) x

run:

M1

M2

M3

M4

5x5=25

Các lớp trừu tượng

- Được sử dụng để xác định ***Cái gì*** các hành vi mà một lớp bắt buộc phải thực hiện mà không cần phải cung cấp một triển khai rõ ràng.
- Nó là kết quả của sự khái quát hóa quá cao
- Cú pháp để xác định một lớp trừu tượng
 - *public abstract class className {...}*
- Tất cả các phương thức trong một lớp trừu tượng là không cần thiết phải trừu tượng.
- Một lớp trừu tượng cũng có thể khai báo các phương thức được triển khai.

es...

```
1 package shapes;
2 public abstract class Shape {
3     abstract public double circumference();
4     abstract public double area();
5 }
6 class Circle extends Shape {
7     double r;
8     public Circle (double rr) { r=rr; }
9     public double circumference() { return 2*Math.PI*r; }
10    public double area() { return Math.PI*r*r; }
11 }
12 class Rect extends Shape {
13     double l,w;
14     public Rect(double ll, double ww) {
15         l = ll; w = ww;
16     }
17     public double circumference() { return 2*(l+w); }
18     public double area() { return l*w; }
19 }
20 class Program {
21     public static void main(String[] args) {
22         Shape s = new Shape ();
23     }
24 }
```

```
20 class Program {
21     public static void main(String[] args) {
22         Shape s = new Circle(5);
23         System.out.println(s.area());
24     }
25 }
```

Đã sửa đổi

Output - Chapter06 (run)

run:
78.53981633974483

Các lớp trừu tượng...

```
1 public abstract class AbstractDemo2 {
2     void m1() // It is not abstract class
3     { System.out.println("m1");
4     }
5     void m2() // It is not abstract class
6     { // empty body
7     }
8     public static void main(String[] args)
9     { AbstractDemo2 obj = new AbstractDemo2();
10    }
11 }
```

Lớp này không có phương thức trừu tượng nhưng nó được khai báo như một lớp trừu tượng. Vì vậy, chúng ta không thể khởi tạo một đối tượng của lớp này.

Các lớp trừu tượng...

Lỗi.

Tại sao?

```
1 public abstract class AbstractDemo2 {
2     void m1() // It is not abstract class
3     { System.out.println("m1");
4     }
5     abstract void m2();
6 }
7
8 class Derived extends AbstractDemo2
9 { public void m1() // override
10    { System.out.println("m1");
11    }
12    public static void main(String[] args)
13    { Derived obj = new Derived();
14    }
```

Thực hiện các phương pháp trừu tượng

- Tạo ra một lớp từ lớp cha trừu tượng, lớp con sẽ kế thừa tất cả các tính năng của lớp cha, tất cả ***trừu tượng*** **phương pháp** bao gồm.
- Để thay thế một phương thức trừu tượng kế thừa bằng một phiên bản cụ thể, lớp con chỉ cần ghi đè lên nó.
- Các lớp trừu tượng ***không thể được khởi tạo***

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2) Abstract class doesn't support multiple inheritance .	Interface supports multiple inheritance .
3) Abstract class can have final, non-final, static and non-static variables .	Interface has only static and final variables .
4) Abstract class can provide the implementation of interface .	Interface can't provide the implementation of abstract class .
6) An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
7) An abstract class can be extended using keyword <code>?extends?</code> .	An interface class can be implemented using keyword <code>?implements?</code> .
8) A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.

Nhập Truyền trong Kế thừa

- Bạn có thể truyền một thể hiện của một lớp con đến lớp cha của nó. Truyền một đối tượng của lớp con sang lớp cha được gọi là **dự báo**.
- Truyền một đối tượng của lớp cha sang lớp con của nó được gọi là **dự báo xuống**.
- **lớp** Phương tiện giao thông { **Sợi dây** nhà chế tạo; }
- **lớp** Xe ô tô **kéo dài** Phương tiện {String typeofEngine; }
- **giai cấp công cộng** TypeCastExample { **công cộng** **tĩnh** void main (**Sợi dây**[] args) {
 Xe v1 = **Mới** Phương tiện giao thông(); Xe v2 = **Mới** Car (); //
 upcasting Car v3 = (Car) **Mới** Xe (); // downcasting Xe v4 =
 Mới Xe ô tô(); }}

Các loại Enum

- Một *loại enum* là một kiểu dữ liệu đặc biệt cho phép một biến trở thành một tập hợp các hằng số được xác định trước.
- Chúng tôi sử dụng các kiểu enum bất cứ lúc nào bạn cần để đại diện cho một tập hợp cố định của các hằng số được đặt tên (chữ hoa).

Mới / Java Enum

```
public enum Day {  
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,  
    THURSDAY, FRIDAY, SATURDAY; // ; can be missed  
}
```

Ví dụ đơn giản về Enum Type

```
enum Mùa {XUÂN, MÙA HÈ, MÙA THU, MÙA ĐÔNG}
```

```
lớp chính {
```

```
    static void fun (Season x)
```

```
    {switch (x)
```

```
        {case SPRING: System.out.println ("Đang là mùa xuân"); nghỉ; case
```

```
        SUMMER: System.out.println ("Đang là mùa hè"); nghỉ; case
```

```
        AUTUMN: System.out.println ("Trời đang mùa thu"); nghỉ; case
```

```
        WINTER: System.out.println ("Đang là mùa đông");
```

```
    }
```

```
    }
```

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

```
        Phần x = Season.WINTER; vui vẽ (x);
```

```
        for (Phần y: Season.values  ()) {
```

```
            System.out.print (y + ":"); vui vẽ (y);
```

```
        }
```

```
        System.out.println ();
```

```
    }
```

```
}
```

```
It is winter
```

```
SPRING: It is spring
```

```
SUMMER: It is summer
```

```
AUTUMN: It is autumn
```

```
WINTER: It is winter
```

Loại Enum với hàm tạo tham số

```
Mùa enum {  
    XUÂN (25, 11), MÙA HÈ (32, 13), MÙA THU (23, 10), MÙA ĐÔNG (10, 9); private  
    final int avgTemp, dayLength;  
    Phần (int x, int y) {  
        avgTemp = x; dayLength = y;  
    }  
    public void display () {  
        System.out.println (this + "nhiệt độ trung bình là" + avgTemp);  
        System.out.println (this + "độ dài trung bình của ngày là" + dayLength);  
    }  
}
```

```
lớp chính {  
    public static void main (String [] args) {  
        Phần x = Season.WINTER;  
        x.display ();  
        System.out.println ();  
    }  
}
```

```
WINTER average temperature is 10  
WINTER average day's length is 9
```


Tóm lược

- Lớp, cách khai báo trường, phương thức và hàm tạo.
- Gợi ý thiết kế lớp học:
 - Danh từ chính → Lớp
 - Danh từ mô tả → Lĩnh vực
 - Phương thức: Constructors, Getters, Setters, Normal method
- Tạo và sử dụng các đối tượng.
- Sử dụng toán tử dấu chấm để truy cập các biến và phương thức thể hiện của đối tượng.

- Ghi đè các phương thức trong lớp con
- Kiểm soát quyền truy cập vào các thành viên của một lớp học bằng cách sử dụng công cụ sửa đổi
- Lớp học lồng nhau
- Lợi ích của việc triển khai OO:
Thừa kế ans Tính đa hình
- Làm việc với các Giao diện.
- Làm việc với các phương thức và lớp trừu tượng.
- Loại Enum