THỪA KẾ
(Inheritance)

Trường ĐH Công nghệ, ĐHQG Hà Nội

Nội dung

- Sử dụng lại trong phát triển phần mềm
- Thừa kế trong Java
  - Quan hệ is-a, has-a (composition)
  - Kiểm soát truy cập, che dấu thông tin
  - Phương thức khởi tạo - constructor
  - Phương thức ghi đè – overriding
  - Thiết kế quan hệ kế thừa
  - Từ khóa final

Tài liệu tham khảo

- *Giáo trình Lập trình HĐT*, Chương 7
- *Java How to Program*, Chương 9
- *Thingking in Java*, Chương 6

Sử dụng lại trong phát triển PM

- Sử dụng lại:
  - Gọi các hàm từ thư viện có sẵn
  - Sử dụng lại các kiến trúc, các thiết kế từ một hệ thống phần mềm trước đó
  - Sửa đổi các module phần mềm tương tự, các mã nguồn từ một hệ thống phần mềm trước đó
  - Copy các mã nguồn (code reuse) tương tự từ các chương trình trước đó
  - Sử dụng lại các lớp trong "object – oriented program"
- Vì sao cần sử dụng lại
  - Hiệu quả kinh tế: giảm chi phí sản xuất, đảm bảo chất lượng tốt

i.e., avoiding double work, reusing good solutions,...

Sử dụng lại trong OOP

- Chương trình có nhiều loại đối tượng có thông tin và hành vi tương tự, liên quan đến nhau:

  Person, Employee, Manager,…

- Nhu cầu tái sử dụng mã nguồn (code reuse)
  - Copy mã nguồn vào chương trình

    Thủ công, dễ nhầm, khó sửa lỗi,…
  - Sử dụng lại thông qua quan hệ "has – a" (composition)

    Chưa mềm dẻo, phải viết lại các giao diện,…
  - Sử dụng lại thông qua quan hệ "is – a" (inheritance)

    Tính đa hình - polymorphism

Quan hệ "has – a"

```
public class Person {
        private String name;
        private Date bithday;
        public String getName() { return name; }
}
```

```
public class Employee {
        private Person me;    //flexible?
        private double salary;
        public String getName() {
            return me.getName(); //flexible?
        }
}
```

```
public class Manager{
        private Employee me;  //flexible?
        private Employee assistant; //flexible?
        public void setAssistant(Employee e) {
            return assistant = e;
        }
}
```

```
…
Manager junior = new Manager();
Manager senior = new Manager();
//a manager can be an assistant of
//another manager
senior.setAssistant(junior);  //error
…
```

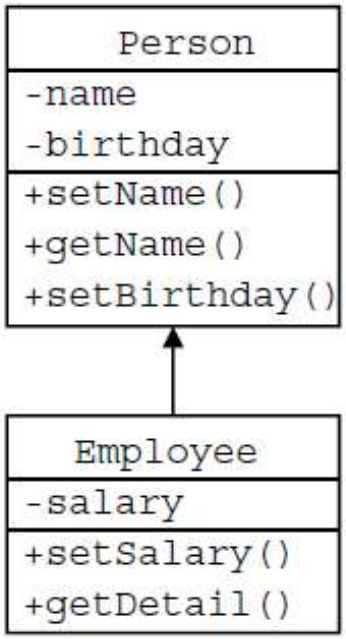
Kế thừa – Inheritance

- Sử dụng lại thông qua quan hệ "is – a"
- Thừa hưởng lại các thuộc tính và phương thức đã có
- Chi tiết hóa cho phù hợp với đối tượng mới
  - Thêm thuộc tính, phương thức mới
  - Hiệu chỉnh, cài đặt chi tiết phương thức được thừa kế
- Lớp cơ sở, lớp cha; lớp dẫn xuất, lớp con

Kế thừa trong Java

```
[public] class DeriveClass extends BaseClass{
}
```

```
public class Person {
        String name;
        Date birthday;
        public void setName(String n) { name = n; }
        public void setBirthday(Date d) { birthday = d; }
        public String getName() { return name; }
}
…
public class Employee extends Person {
        …
        double salary;
        public boolean setSalary (double sal) {
                salary = sal;
                return true;
        }
        …
}
```




```
Employee e = new Employee ();
e.setName("John");
e.setSalary(3.66);
System.out.print (e.getDetail());
```

Kiểm soát truy cập

- public, protected, “default” (packaged), private

	Same Class	Package	Subclass (same package)	Subclass (different package)	Universal
public	+	+	+	+	+
protected	+	+	+	+	
packaged	+	+			
private	+				

Mức truy cập protected

- Đối tượng thuộc lớp dẫn xuất truy cập được các *protected members* (*thuộc tính, phương thức*) *của lớp cơ sở*
- Với các đối tượng khác *che giấu thông tin*

```
public class Person {
          protected String name;
          protected int age;
          …
}
public class Employee extends Person {
          double salary;
          public String getDetail () {
             String s = name + “, ”+ age;
             s += “, “ + salary;
             return s;
          }
          …
}
```

Private trong kế thừa

- Lớp dẫn xuất không kế thừa các *private members* của lớp cơ sở

thành phần bị ẩn

- Có thể truy cập các *thành phần private (bị ẩn)* thông qua các giao diện của lớp cơ sở

Đối tượng của lớp dẫn xuất có chứa thông tin thuộc tính private?

```
public class Person {
   private String name;
   private int age;
   public String getNam(){
            return name;}
   public int getAge () {
            return age;}
            …
}
public class Employee extends Person {
    double salary;
    public String getDetail () {
        //String s = name + “, ”+ age;
        String s = getNam() + “, “+ getAge();
        s += “, “ + salary;
        return s;
    }
            …
}
```

Kế thừa: trong cùng gói, khác gói

Trong cùng gói

```
class Person {
          String name;
          int age;
          public String getNam(){
                    return name;}
          public int getAge () {
                    return age;}
          …
}
class Employee extends Person {
   double salary;
   …
   public String getDetail () {
      String s = name + “, ” + age;
      s += “, “ + salary;
      return s;
   }
          …
}
```

Khác gói

```
package comp;
public class Person {
          protected String name;
          protected int age;
          public String getNam(){
                    return name;}
          public int getAge () {
                    return age;}
          …
}
import comp.Person;
public class Employee extends Person {
   double salary;
   …
   public String getDetail () {
      String s = name + “, ” + age;
      s += “, “ + salary;
      return s;
   }
          …
}
```

Kế thừa từ gói khác

- Có thể kế thừa từ gói khác
  - Kế thừa thư viện của Java: ví dụ từ Applet
  - Kế thừa từ gói của các nhà phát triển khác
- Kế thừa mà không cần biết mã nguồn
  - Bảo mật mã nguồn
  - Nâng cao khả năng sử dụng lại

Kế thừa: Phương thức khởi tạo

- Lớp dẫn xuất *không kế thừa* các phương thức khởi tạo (*constr*)
- Lớp dẫn xuất:
  - Gọi constr mặc định của nó hoặc gọi constr chính lớp đó xây dựng

```
public class Point {
   protected int x, y;
   public Point (){}
   public Point (int xx, int yy){
            x = xx;
            y = yy;
   }
…
}
public class Circle extends Point {
   protected double radius;
   public Circle () {}
}
…
Point p1 = new Point (10, 10);
Circle c1 = new Circle ();
Circle c2 = new Circle (10, 10); //error
```

Lớp dẫn xuất: Phương thức khởi tạo

- Phương thức khởi tạo của lớp dẫn xuất:
  - Lớp cơ sở có phương thức khởi tạo (constr) mặc định: có thể gọi hoặc không gọi constr của lớp cơ sở
  - Gọi constr của lớp cơ sở thông qua từ khóa super

```
public class Point {
   protected int x, y;
   public Point (){}
   public Point (int xx, int yy){
            x = xx;
            y = yy;
   }
…
}
public class Circle extends Point {
   protected double radius;
   public Circle () {}
   public Circle (int xx, int yy, double r){
            super (xx, yy); //có thể bỏ câu lệnh này
            radius = r;
   }
}
```

Lớp dẫn xuất: Phương thức khởi tạo

- Phương thức khởi tạo của lớp dẫn xuất:
  - Lớp cơ sở không có constr mặc định: bắt buộc phải gọi constr của lớp cơ sở một cách tường minh
  - Câu lệnh super gọi constr của lớp cơ sở được đặt đầu tiên

```
public class Point {
   protected int x, y;
   public Point (int xx, int yy){
            x = xx;
            y = yy;
   }
…
}
public class Circle extends Point {
   protected double radius;
   public Circle () { super (0,0); }
   public Circle (int xx, int yy, double r){
            super (xx, yy); //bắt buộc phải gọi
            radius = r;
   }
}
```

Lớp dẫn xuất: Phương thức khởi tạo

- Thứ tự khởi tạo trong constr của lớp dẫn xuất:
  - Constr của lớp cơ sở được gọi
  - Các lệnh của constr lớp dẫn xuất được gọi tiếp theo

```
public class Point {
   protected int x, y;
   public Point (){
        System.out.println("Point constructor");
   }
…
}
public class Circle extends Point {
   protected double radius;
   public Circle (){
         System.out.println("Circle constructor");
   }
}
…
Circle c = new Circle ();

---------------------------------
Point constructor
Circle constructor
```

Định nghĩa lại - overriding

- Có thể định nghĩa lại các phương thức được kế thừa (non-private) từ lớp cơ sở:
  - Chi tiết hóa, phù hợp với lớp dẫn xuất
- Khi gọi một phương thức của lớp dẫn xuất:
  - Thực hiện theo cơ chế liên kết động (dynamic binding)
  - Nếu lớp dẫn xuất định nghĩa lại phương thức, phiên bản định nghĩa lại phương thức được thực hiện
  - Nếu lớp dẫn xuất không định nghĩa lại phương thức đó, phương thức được định nghĩa từ lớp cơ sở "gần nhất" sẽ được gọi (bottom – up binding)

Định nghĩa lại - overriding

- Có thể gọi phương thức của lớp cha bằng từ khóa super

```
class Person {
        String name;
        int age;
        public String getDetail () {…}
        …
}
class Employee extends Person {
   double salary;
   …
   public String getDetail () {
      String s;
      s = super.getDetail() +“, “ + salary;
      return s;
   }
}
```

- Gọi phương thức của lớp cơ sở cao hơn (lớp ông, …) như thế nào?
  - super.super…?
  - Thông qua gọi phương thức của lớp cha

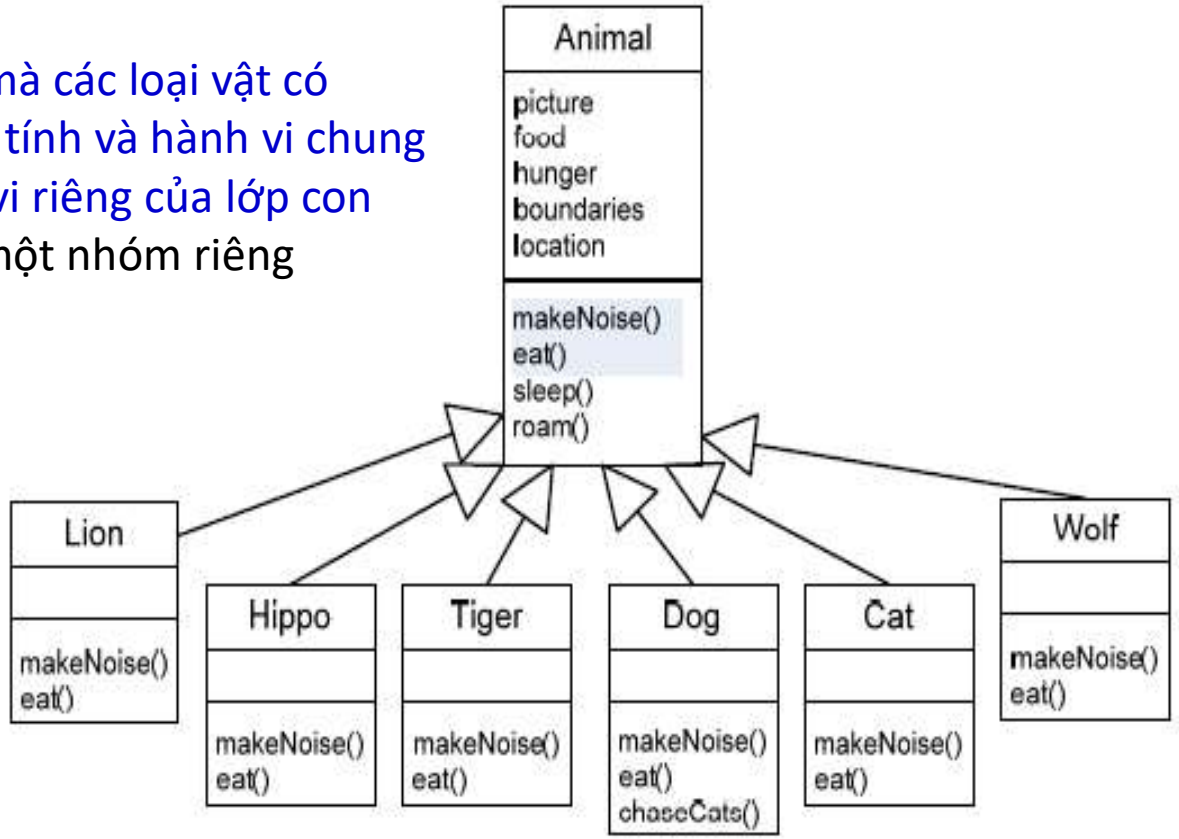
Định nghĩa lại - overriding

- Phạm vi truy cập
  - Phương thức được định nghĩa lại có phạm vi truy cập không chặt hơn phạm vị truy cập của lớp cơ sở
- Kiểu giá trị trả lại như nhau trong phương thức được định nghĩa lại
- Phương thức private
  - Không được kế thừa ở lớp dẫn xuất
  - Không được coi là "định nghĩa lại"

```
public class Parent{
   public void methodA() {}
   private int methodB() {
            return 0;
   }
…
}
public class Child extends Parent {
   protected void methodA() {} //error
   private void methodB() {}
}
```

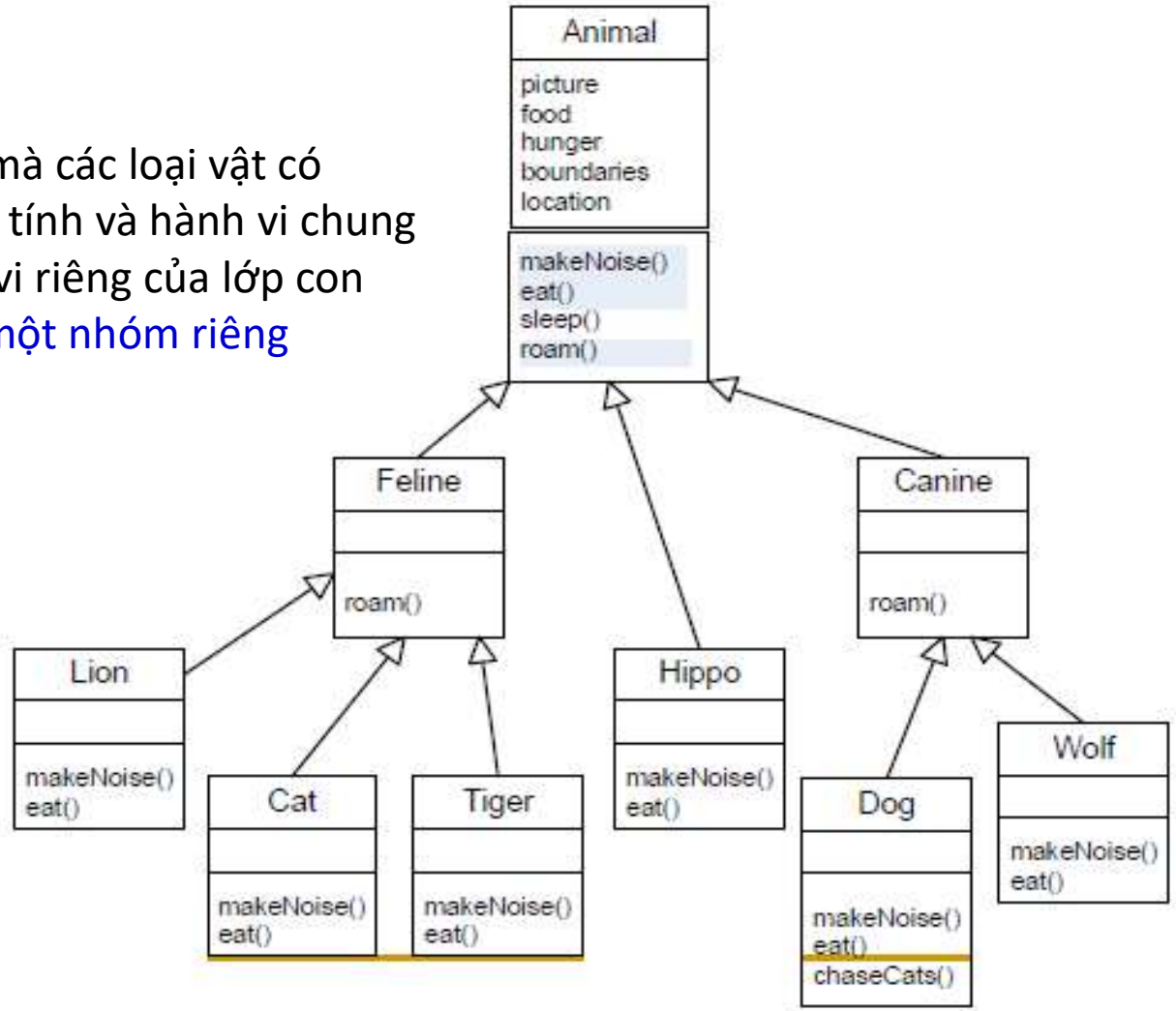
Thiết kế quan hệ kế thừa

1. Tìm tất cả các đặc trưng chung mà các loại vật có
2. Thiết kế lớp cha với tất cả thuộc tính và hành vi chung
3. Thiết kế các thuộc tính và hành vi riêng của lớp con
4. Nhóm tập nhỏ các lớp con vào một nhóm riêng nếu có thể.



Thiết kế quan hệ kế thừa

1. Tìm tất cả các đặc trưng chung mà các loại vật có
2. Thiết kế lớp cha với tất cả thuộc tính và hành vi chung
3. Thiết kế các thuộc tính và hành vi riêng của lớp con
4. Nhóm tập nhỏ các lớp con vào một nhóm riêng nếu có thể.



Từ khóa final

- Thuộc tính final
  - Hằng số, gán giá trị một lần không thay đổi được.

→ không cho thay đổi giá trị của biến

- Phương thức final
  - Không cho phép định nghĩa lại ở lớp dẫn xuất

→ không cho ghi đè (overriding) phương thức

- Tham số final
  - Tham số truyền vào không gán thay đổi giá trị

→ nhưng cho phép thay đổi giá trị thuộc tính của tham chiếu đối tượng

- Lớp final
  - Không định nghĩa được lớp dẫn xuất

→ lớp final không cho phép có lớp con

Tham số final

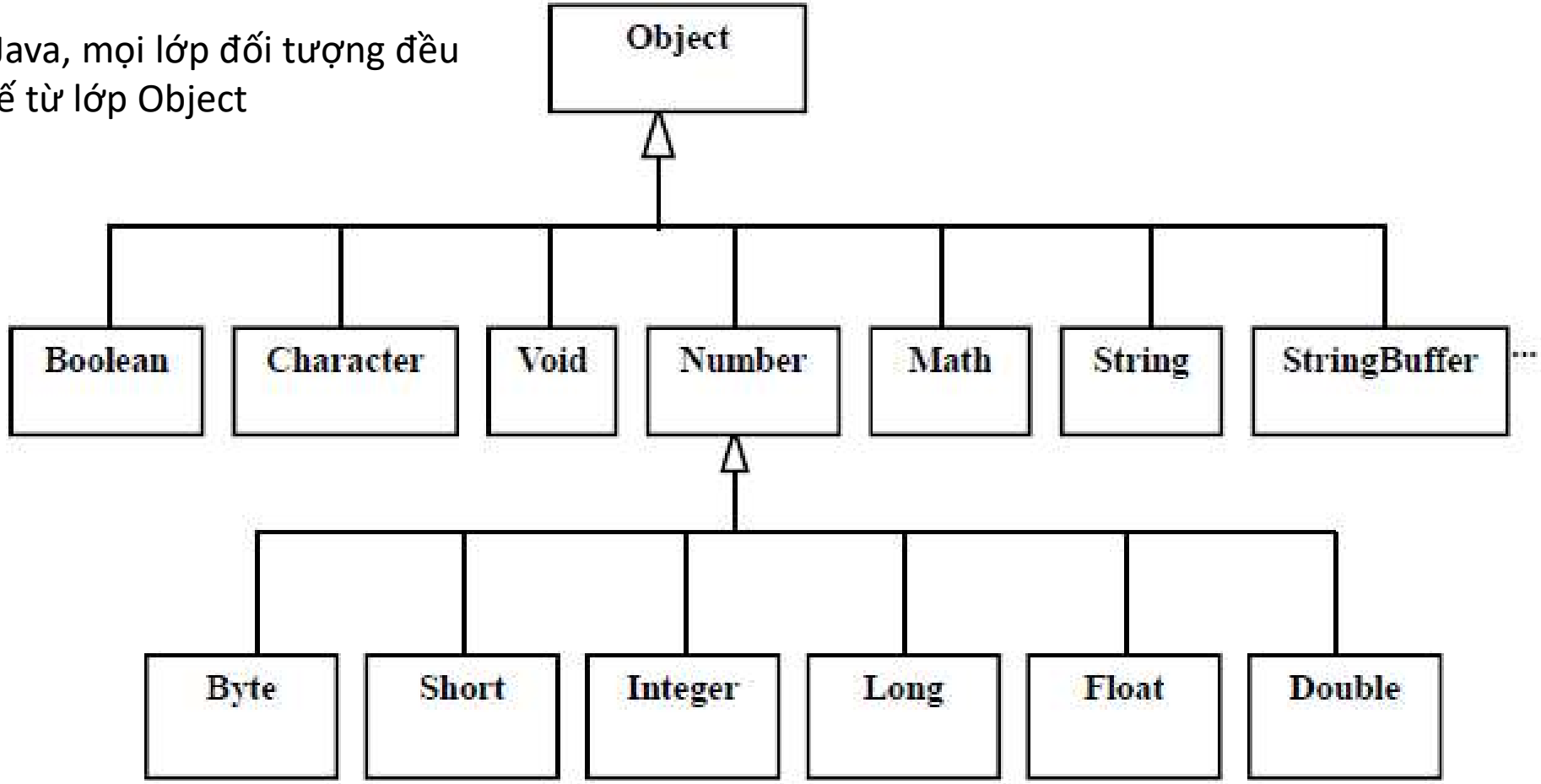
```
class MyDate {
        int year, month, day;
        public MyDate(int y, int m, int d) {
            year = y;
            month = m;
            day = d;
        }
        public void copyTo (final MyDate d) {
            d.year = year;
            d.month = month;
            d.day = day;
            //d = new MyDate(year, month, day);
        }
…
}
```

Tham số truyền vào không gán thay đổi giá trị

→ nhưng cho phép thay đổi giá trị thuộc tính của tham chiếu đối tượng

Lớp Object

Trong Java, mọi lớp đối tượng đều thừa kế từ lớp Object



Thanks for your attention