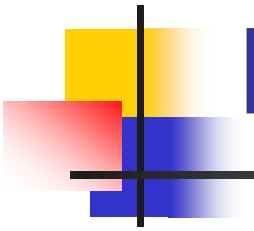
Bộ môn Công nghệ Phần mềm
Viện CNTT & TT
Trường Đại học Bách Khoa Hà Nội

IT3100

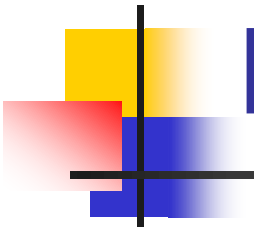
LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Bài 08. Đa hình



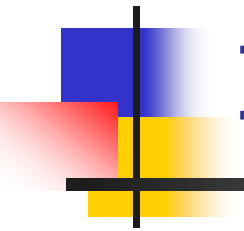
Mục tiêu

- Giới thiệu về upcasting và downcasting
- Phân biệt liên kết tĩnh và liên kết động
- Nắm vững kỹ thuật đa hình
- Ví dụ và bài tập về các vấn đề trên với ngôn ngữ lập trình Java



Nội dung

1. Upcasting và Downcasting
2. Liên kết tĩnh và Liên kết động
3. Đa hình (Polymorphism)
4. Ví dụ và bài tập



1. Upcasting và Downcasting

1. Upcasting và Downcasting

- Chuyển đổi kiểu dữ liệu nguyên thủy
  - Java tự động chuyển đổi kiểu khi
    - Kiểu dữ liệu tương thích
    - Chuyển đổi từ kiểu hẹp hơn sang kiểu rộng hơn

```
int i;
double d = i;
```

  - Phải ép kiểu khi
    - Kiểu dữ liệu tương thích
    - Chuyển đổi từ kiểu rộng hơn sang kiểu hẹp hơn

```
int i;
byte b = i;   byte b = (byte)i;
```

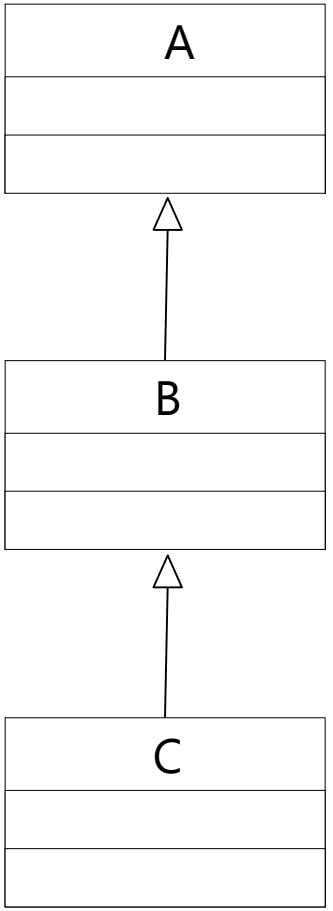
1. Upcasting và Downcasting

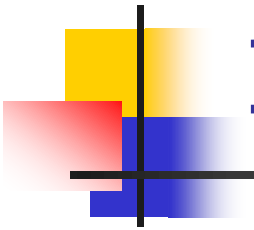
- Chuyển đổi kiểu dữ liệu tham chiếu
  - Kiểu dữ liệu tham chiếu có thể được chuyển đổi kiểu khi
    - Kiểu dữ liệu tham chiếu (lớp) *tương thích*
      - *Nằm trên cùng một cây phân cấp kế thừa*

```
A var1 = new B();

A var1 = new A();
C var2 = (C)var1;
```

  - Hai cách chuyển đổi
    - Up-casting
    - Down-casting

"*dán cho nó một cái nhãn*"





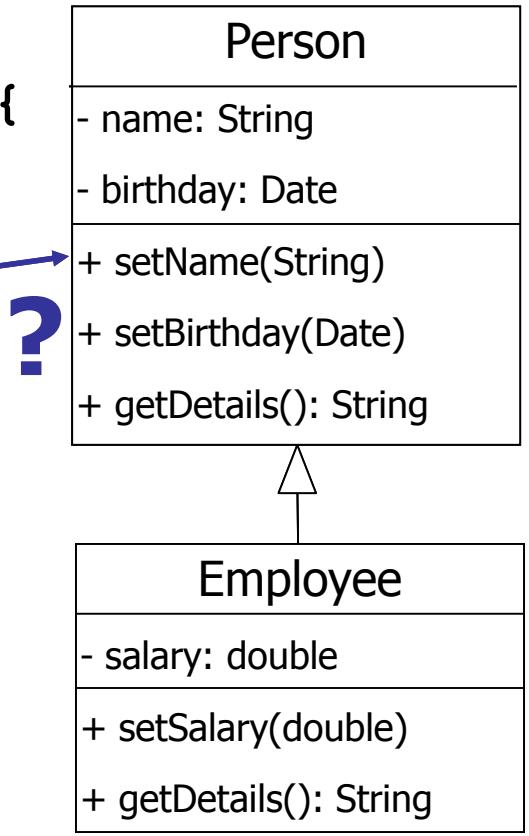
1.1 Upcasting

- Up casting: đi lên trên cây phân cấp thừa kế (moving up the inheritance hierarchy)
- Up casting là khả năng nhìn nhận đối tượng thuộc lớp dẫn xuất như là một đối tượng thuộc lớp cơ sở.
- Tự động chuyển đổi kiểu

1.1 Upcasting

- Ví dụ:

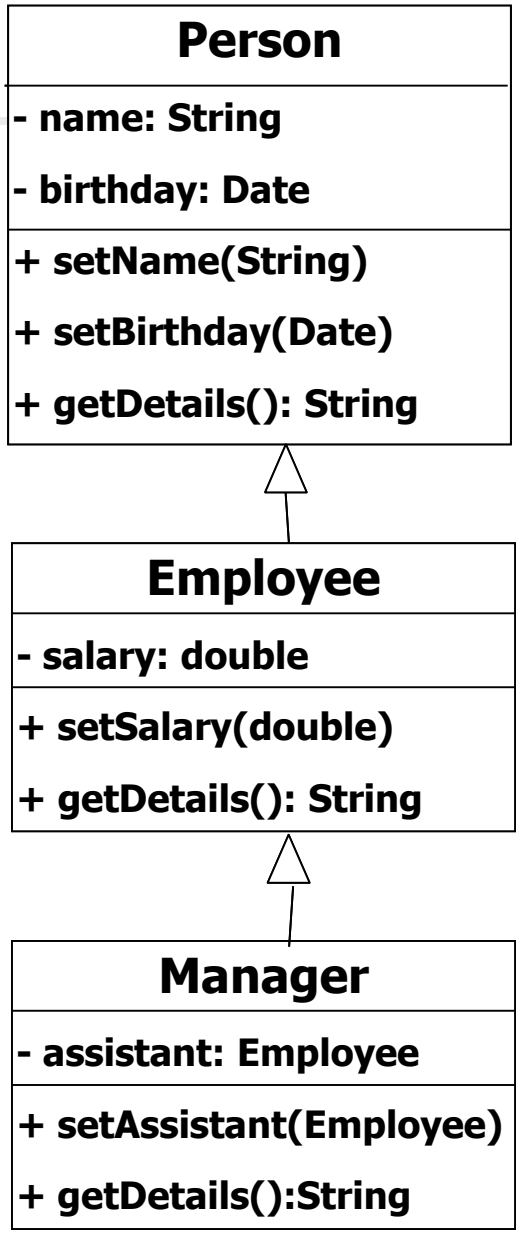
```
public class Test1 {
   public static void main(String arg[]) {
        Employee e = new Employee();
        Person p;
        p = e;
        p.setName("Hoa");
        p.setSalary(350000);
          // compile error
}
```



1.1 Upcasting

- Ví dụ:

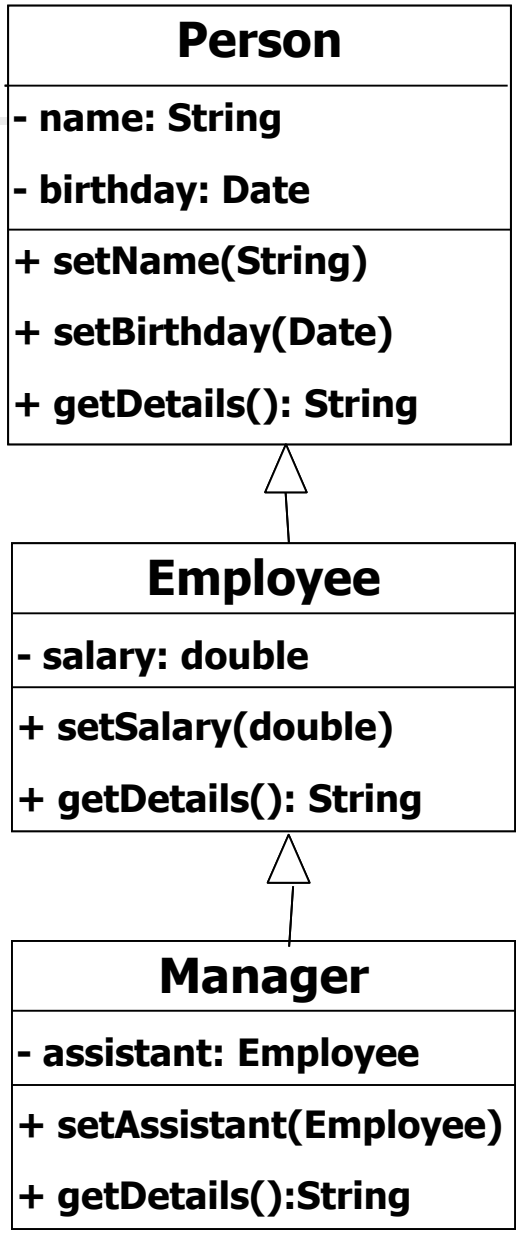
```
class Manager extends Employee {
   Employee assistant;
   // ...
   public void setAssistant(Employee e) {
        assistant = e;
   }
   // ...
}
public class Test2 {
   public static void main(String arg[]) {
        Manager junior, senior;
        // ...
        senior.setAssistant(junior);
   }
}
```



1.1 Upcasting

- Ví dụ:

```
public class Test3 {
   String static teamInfo(Person p1,
                          Person p2) {
      return "Leader: " + p1.getName() +
             ", member: " + p2.getName();
   }
   public static void main(String arg[]) {
      Employee e1, e2;
      Manager m1, m2;
      // ...
      System.out.println(teamInfo(e1, e2));
      System.out.println(teamInfo(m1, m2));
      System.out.println(teamInfo(m1, e2));
   }
}
```



1.2 Downcasting

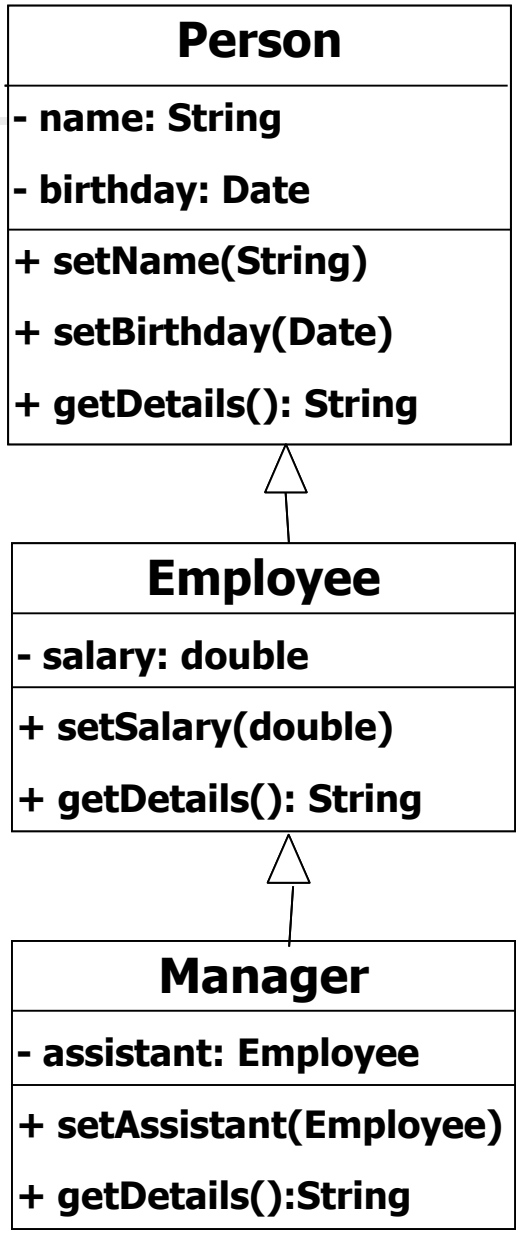
- Down casting: đi xuống cây phân cấp thừa kế (move back down the inheritance hierarchy)
- Down casting là khả năng nhìn nhận một đối tượng thuộc lớp cơ sở như một đối tượng thuộc lớp dẫn xuất.
- Không tự động chuyển đổi kiểu

  → Phải ép kiểu.

1.2 Downcasting

- Ví dụ:

```
public class Test2 {
   public static void main(String arg[]) {
        Employee e = new Employee();
        Person p = e; // up casting
        Employee e1 = (Employee) p;
        // down casting
        Manager m = (Manager) ee;
        // run-time error

        Person p2 = new Manager();
        Employee e2 = (Employee) p2;
   }
}
```



Toán tử `instanceof`

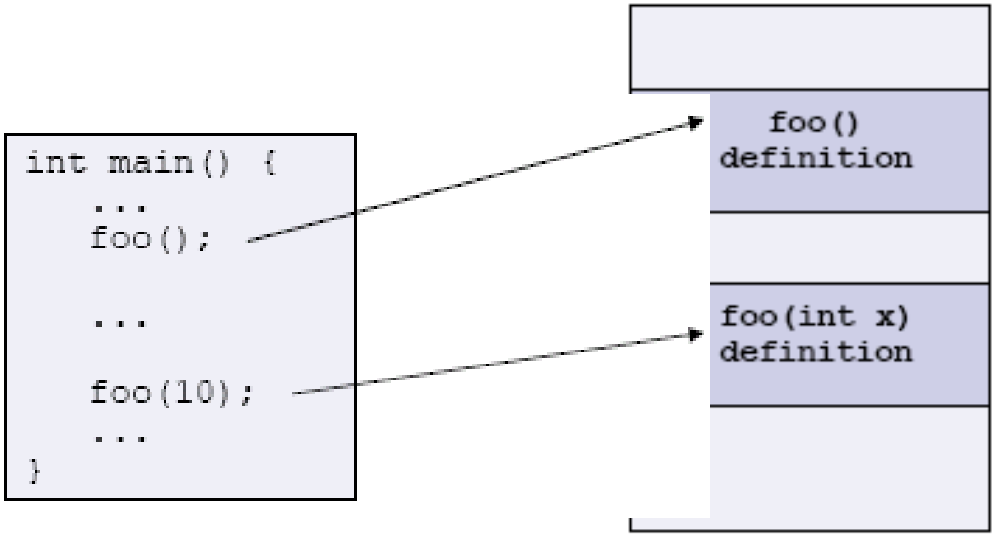
- Kiểm tra xem một đối tượng có phải là thể hiện của một lớp nào đó không
- Trả về: true | false (nếu đối tượng là null thì trả về false)

```
public class Employee extends Person {}
public class Student extends Person {}

public class Test{
  public doSomething(Person e) {
    if (e instanceof Employee) {...
    } else if (e instanceof Student) {...
    } else {...
    }
  }
}
```

2. Liên kết tĩnh và liên kết động (Static binding & dynamic binding)

Liên kết lời gọi hàm

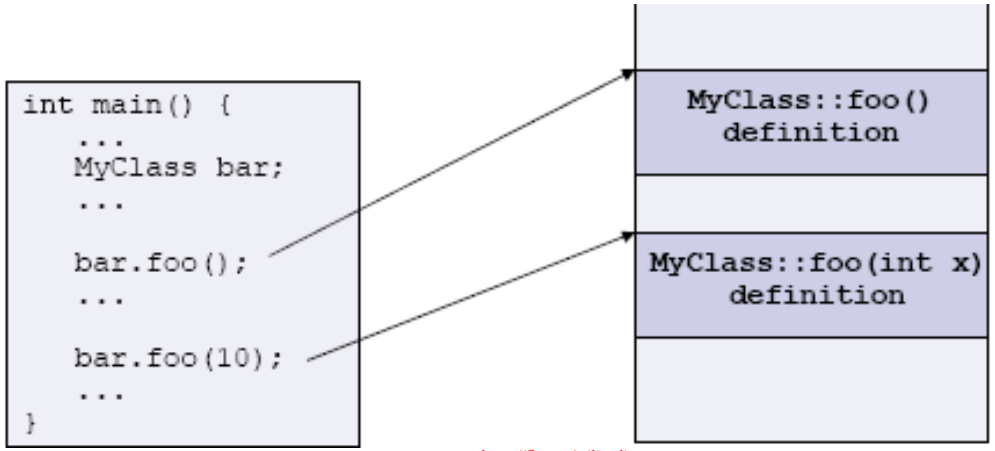
- Liên kết lời gọi hàm (function call binding) là quy trình xác định khối mã hàm cần chạy khi một lời gọi hàm được thực hiện
  - C: đơn giản vì mỗi hàm có duy nhất một tên
  - C++: chồng hàm, phân tích chữ ký kiểm tra danh sách tham số



Trong ngôn ngữ Hướng đối tượng

- Liên kết lời gọi phương thức
- Đối với các lớp độc lập (không thuộc cây thừa kế nào), quy trình này gần như không khác với function call binding
  - so sánh tên phương thức, danh sách tham số để tìm định nghĩa tương ứng
  - một trong số các tham số là tham số ẩn: con trỏ this

`bar.foo ();` → lời gọi này bị ràng buộc với định nghĩa của phương thức mà nó gọi



2.1 Liên kết tĩnh

- Liên kết tại thời điểm biên dịch
  - Early Binding/Compile-time Binding
  - Lời gọi phương thức được quyết định khi biên dịch, do đó chỉ có một phiên bản của phương thức được thực hiện
  - Nếu có lỗi thì sẽ có lỗi biên dịch
  - Ưu điểm về tốc độ
- C/C++ function call binding, và C++ method binding cơ bản đều là ví dụ của liên kết tĩnh (static function call binding)

2.1 Liên kết tĩnh

- Thích hợp cho các lời gọi hàm thông thường
  - Mỗi lời gọi hàm chỉ xác định duy nhất một định nghĩa hàm, kể cả trường hợp hàm chồng.
- Phù hợp với các lớp độc lập không thuộc cây thừa kế nào
  - Mỗi lời gọi phương thức từ một đối tượng của lớp hay từ con trỏ đến đối tượng đều xác định duy nhất một phương thức

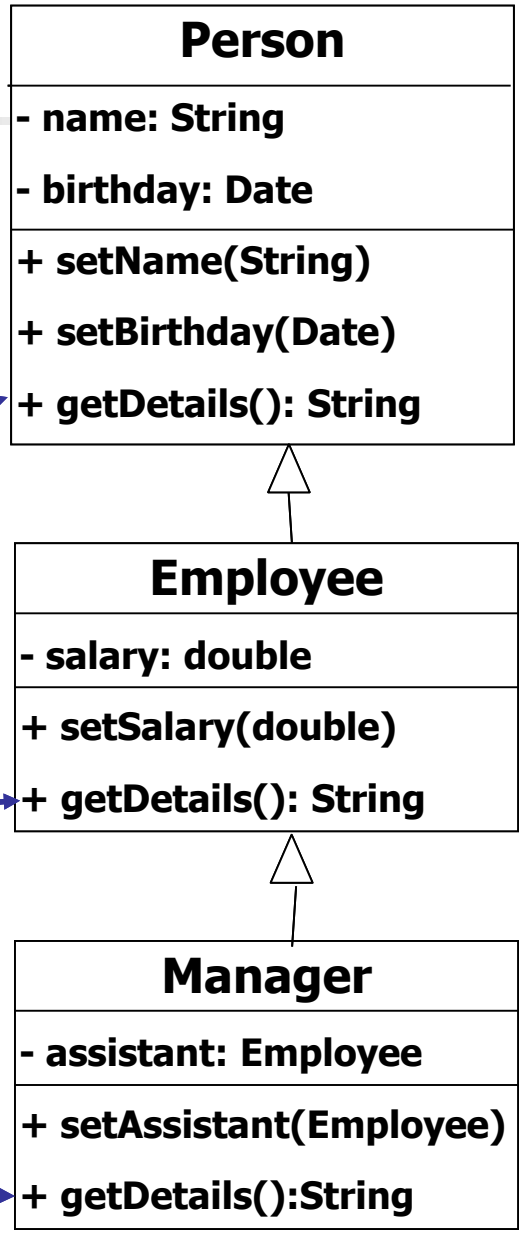
2.2 Liên kết động

- Lời gọi phương thức được quyết định khi thực hiện (run-time)
  - Late binding/Run-time binding
  - Phiên bản của phương thức phù hợp với đối tượng được gọi
  - Java trì hoãn liên kết phương thức cho đến thời gian chạy (run-time) - đây được gọi là liên kết động hoặc liên kết trễ
    - Java mặc định sử dụng liên kết động

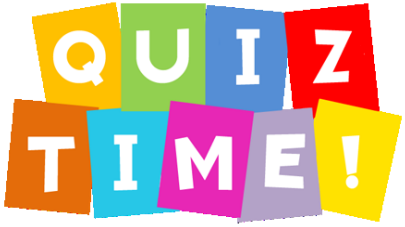
Ví dụ

```
public class Test {
  public static void main(String arg[]){
     Person p = new Person();
     // ...
     Employee e = new Employee();
     // ...
     Manager m = new Manager();
     // ...
     Person pArr[] = {p, e, m};
     for (int i=0; i< pArr.length; i++){
       System.out.println(
             pArr[i].getDetail());
     }
   }
}
```

Tuỳ thuộc vào đối tượng gọi tại thời điểm thực thi chương trình (run-time)



Câu hỏi

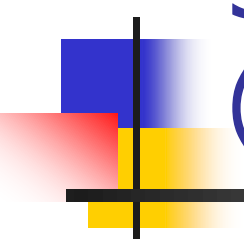


- Giả sử lớp Sub kế thừa từ lớp cha Sandwich. Tạo hai đối tượng từ các lớp này:

```
Sandwich x = new Sandwich();
Sub y = new Sub();
```

- Phép gán nào sau đây là hợp lệ?

```
1. x = y;
2. y = x;
3. y = new Sandwich();
4. x = new Sub();
```

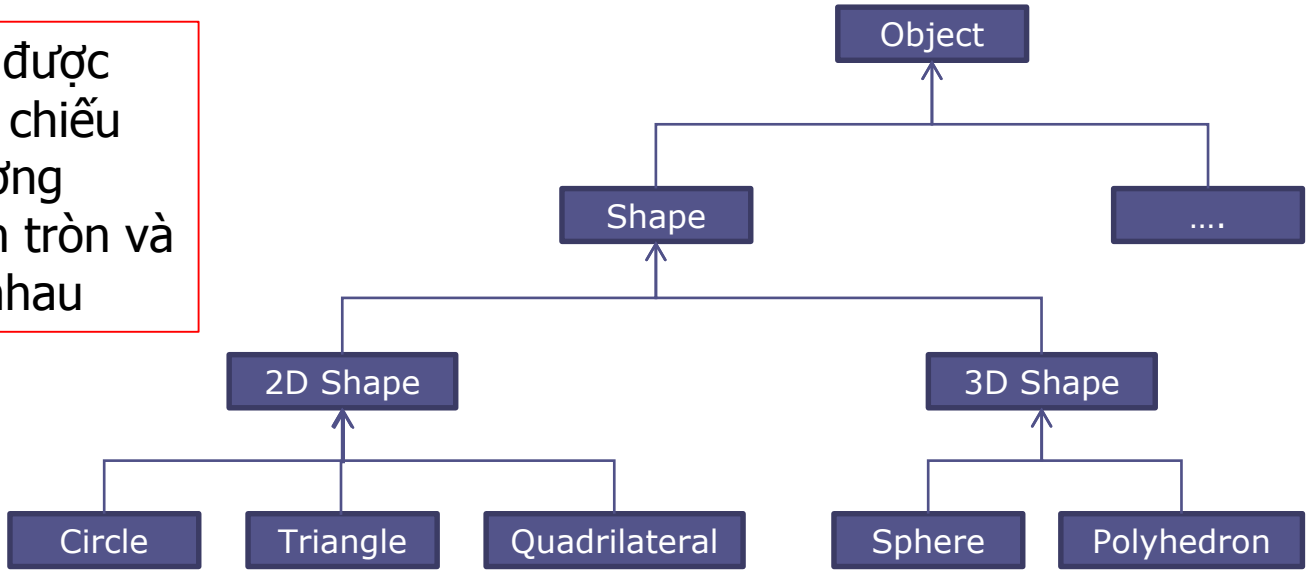


3. Đa hình (Polymorphism)

3. Đa hình

- Ví dụ: Một hoạt động có thể được thực hiện trên một đối tượng 2DShape cũng có thể được thực hiện trên một đối tượng thuộc một trong ba lớp Tam giác, Hình tròn, Tứ giác.
  - Lớp cha 2DShape định nghĩa giao diện chung
  - Các lớp con Tam giác, Vòng tròn, Tứ giác phải theo giao diện này (kế thừa), nhưng cũng được phép cung cấp các triển khai riêng của chúng (ghi đè)

→ Khi một phương thức được yêu cầu thông qua tham chiếu lớp 2DShape, các đối tượng 2DShape, Tam giác, Hình tròn và Tứ giác phản ứng khác nhau



3. Đa hình

- Ví dụ:

```
public class 2DShape {
      public void display() {
          System.out.println("2D Shape");
      }
}
```

```
public class Point extends 2DShape {
      private int x, y;
      ...
      public void display(){
            System.out.print("(" + x + "," + y + ")");
      }
}
```

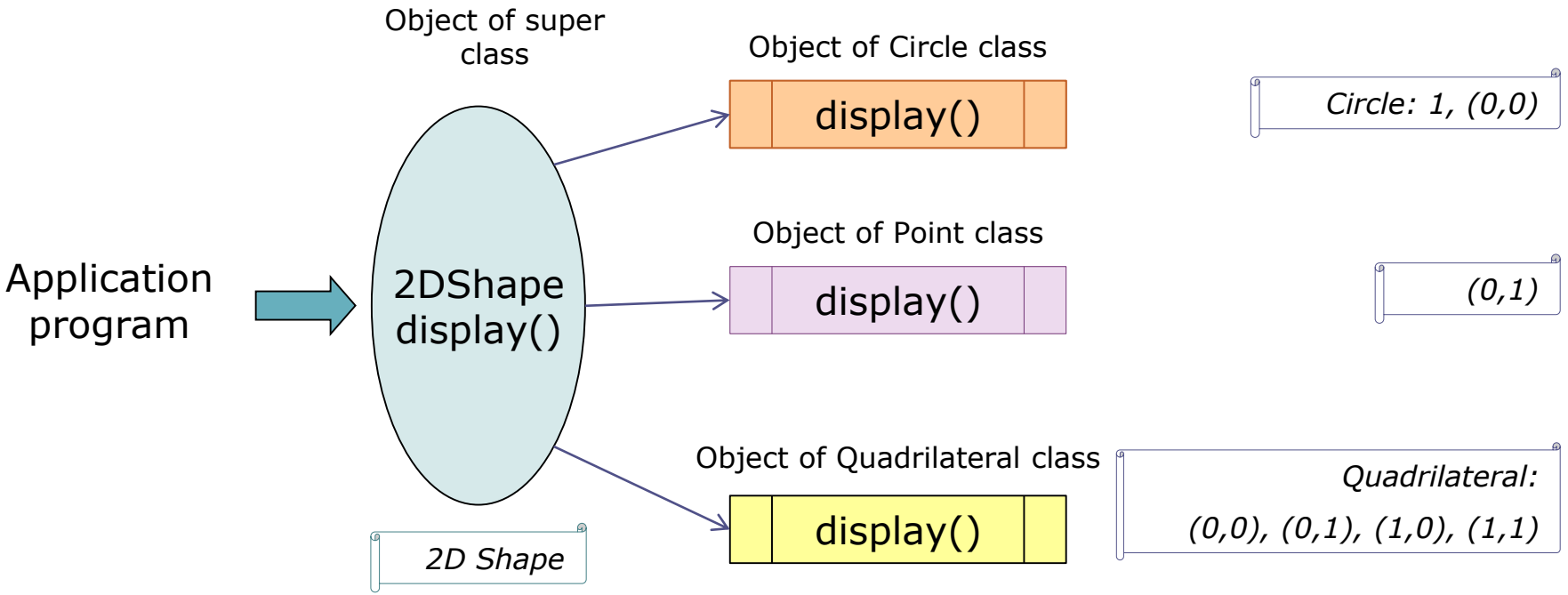
```
public class Circle extends 2DShape{
      public static final double PI =
          3.14159;
      private Point p;
      private double r; //radious

      ...
      public void display(){
            System.out.print("Circle: " +
              r + ",");
            p.display();
            System.out.println();
      }
}
```



```
public class Quadrilateral extends 2DShape {
      private Point p1, p2, p3, p4;

      .....

      public void display(){
            System.out.println("Quadrilateral: ");
            p1.display(); p2.display();
            p3.display(); p4.display();
            System.out.println();
      }
}
```

3. Đa hình

- Ví dụ: Có nhiều sự lựa chọn một khi một phương thức được gọi thông qua một tham chiếu lớp cha.



3. Đa hình

- Polymorphism: Nhiều hình thức thực hiện, nhiều kiểu tồn tại
  - Khả năng của một biến tham chiếu thay đổi hành vi theo đối tượng mà nó đang giữ.
- Đa hình trong lập trình
  - Đa hình phương thức:
    - Phương thức trùng tên, phân biệt bởi danh sách tham số.
  - Đa hình đối tượng
    - Nhìn nhận đối tượng theo nhiều kiểu khác nhau
    - Các đối tượng khác nhau cùng đáp ứng chung danh sách các thông điệp có giải nghĩa thông điệp theo cách thức khác nhau.

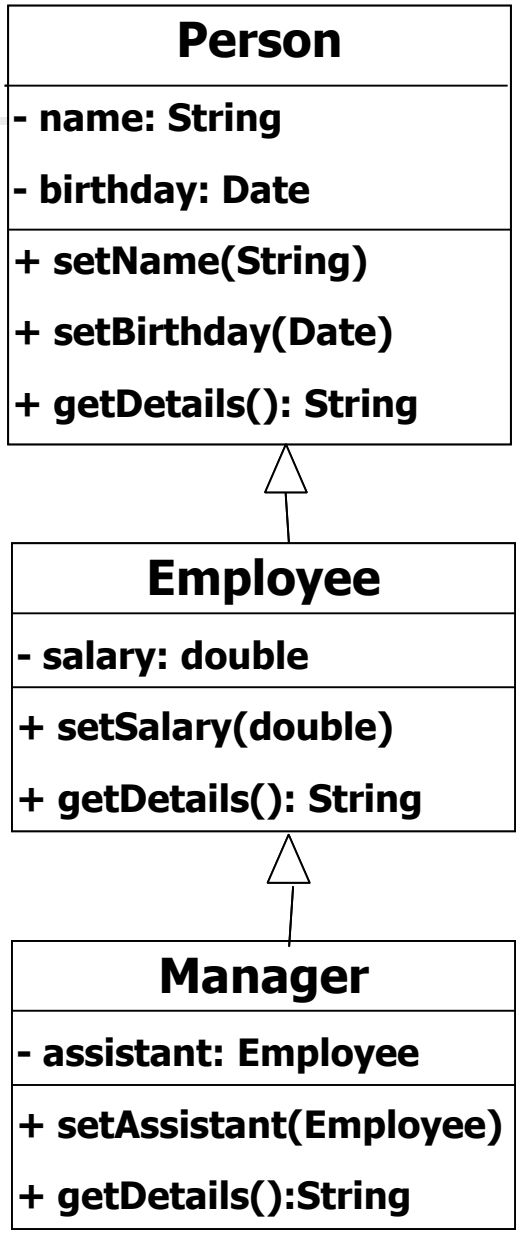
3. Đa hình

- Polymorphism: gia tăng khả năng tái sử dụng những đoạn mã nguồn được viết một cách tổng quát và có thể thay đổi cách ứng xử một cách linh hoạt tùy theo loại đối tượng
  - Tính đa hình (*Polymorphism*) trong Java được hiểu là trong từng trường hợp, hoàn cảnh khác nhau thì đối tượng có hình thái khác nhau tùy thuộc vào từng ngữ cảnh
- Để thể hiện tính đa hình:
  - Các lớp phải có quan hệ kế thừa với 1 lớp cha nào đó
  - Phương thức được ghi đè (override) ở lớp con

3. Đa hình

- Ví dụ:
- Các đối tượng khác nhau giải nghĩa các thông điệp theo các cách thức khác nhau
  - Liên kết động (Java)

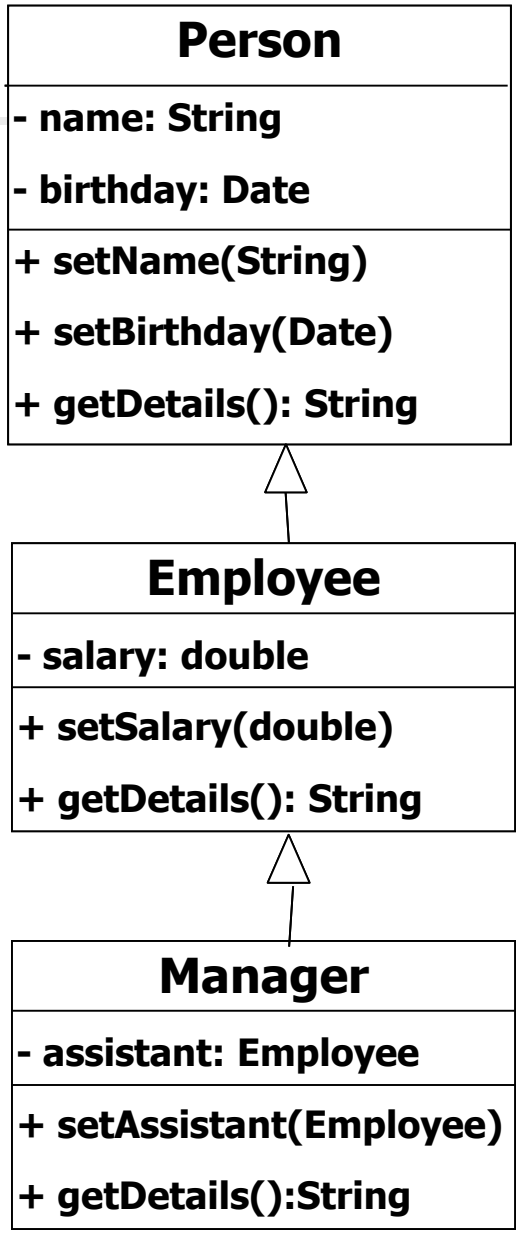
```
Person p1 = new Person();
Person p2 = new Employee();
Person p3 = new Manager();
// ...
System.out.println(p1.getDetail());
System.out.println(p2.getDetail());
System.out.println(p3.getDetail());
```



3. Đa hình

- Ví dụ:

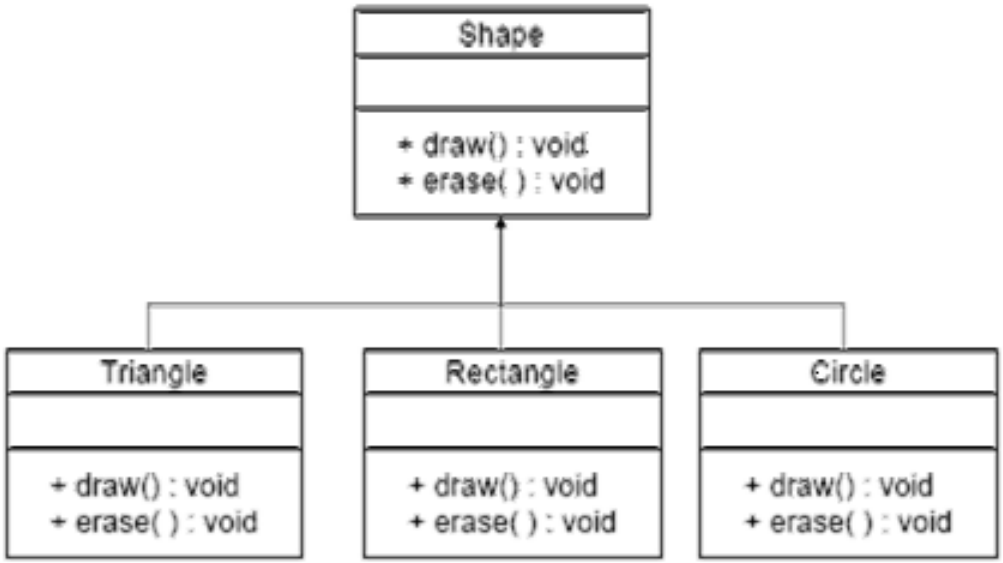
```
class EmployeeList {
     Employee list[];
     ...
     public void add(Employee e) {...}
     public void print() {
        for (int i=0; i<list.length; i++) {
              System.out.println(list[i].
                                    getDetail());
        }
     }
    ...

     EmployeeList list = new EmployeeList();
     Employee e1; Manager m1;
     ...
     list.add(e1);
     list.add(m1);
     list.print();
```



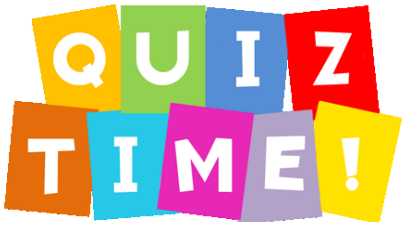
3. Đa hình

- Ví dụ: Các đối tượng Triangle, Rectangle, Circle đều là các đối tượng Shape

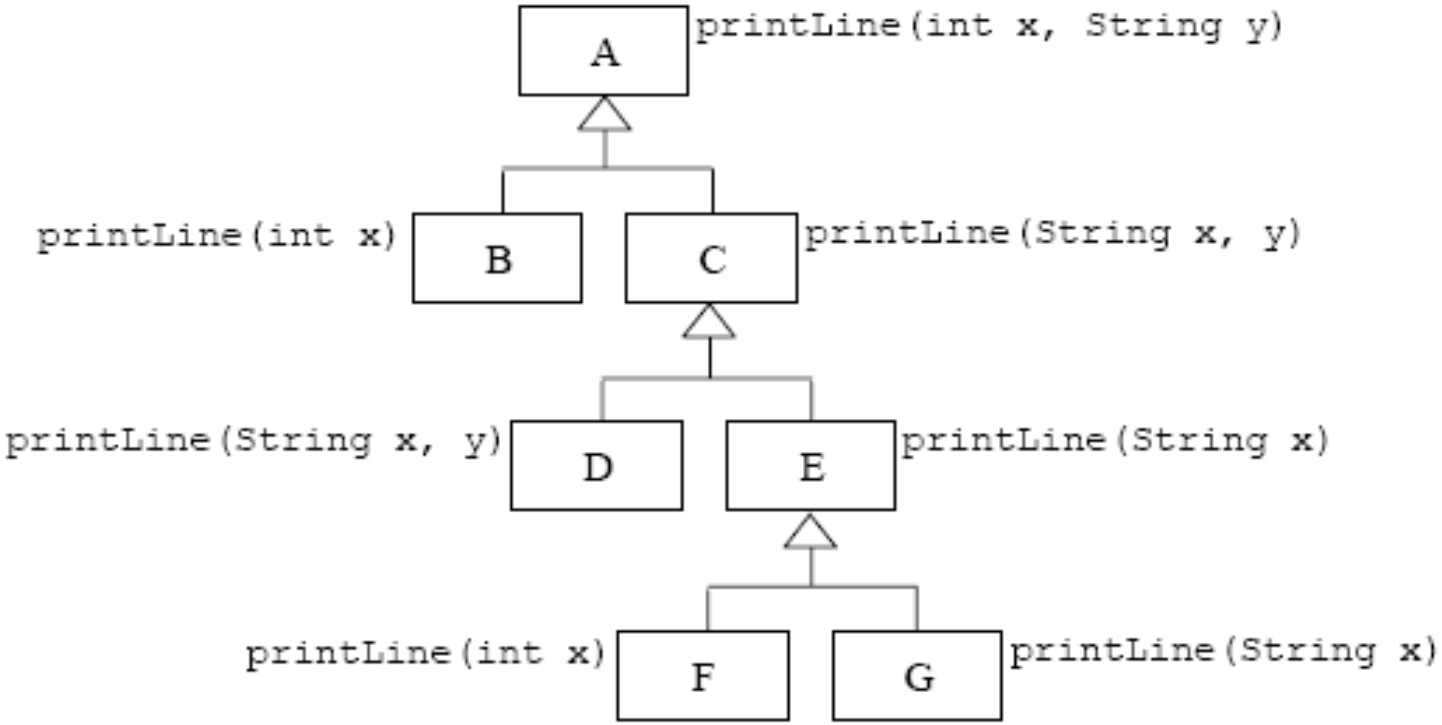
```
...
public static void handleShapes(Shape[] shapes){
  // Vẽ các hình theo cách riêng của mỗi hình
  for( int i = 0; i < shapes.length; ++i) {
    shapes[i].draw();
  }
  ...
  // Gọi đến phương thức xóa
  // không cần quan tâm đó là
  for( int i = 0; i < shapes
    shapes[i].erase();
  }
}
...
```



Câu hỏi



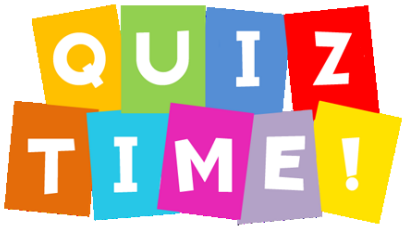
- Cho biểu đồ lớp:



Phương thức **`printLine()`** của lớp nào sẽ được sử dụng trong mỗi trường hợp dưới đây, biết rằng **z** là một đối tượng của lớp **F**? Giải thích ngắn gọn?

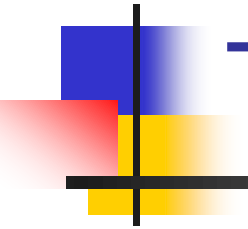
```
1. z.printLine(1)
2. z.printLine(2, "Object-Oriented Programming")
3. z.printLine("Java")
4. z.printLine("Object-Oriented Programming", "Java")
5. z.printLine("Object-Oriented Programming", 3)
```

Câu hỏi

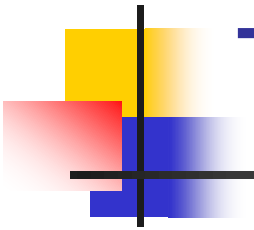


- Những điều kiện nào trả về true? (Có thể xem Java documentation để biết các quan hệ thừa kế giữa các lớp) Biết rằng System.out là một đối tượng của lớp PrintStream.

```
1. System.out instanceof PrintStream
2. System.out instanceof OutputStream
3. System.out instanceof LogStream
4. System.out instanceof Object
5. System.out instanceof String
6. System.out instanceof Writer
```

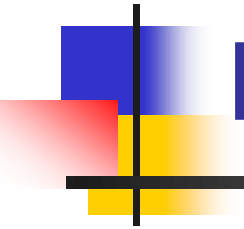


Tổng kết



Tổng kết

- Upcasting và downcasting
  - Nhìn nhận các đối tượng thuộc lớp cơ sở như đối tượng thuộc lớp dẫn xuất (upcasting) và ngược lại (down-casting)
- Liên kết tĩnh và liên kết động
  - Liên kết lời gọi hàm lúc biên dịch (liên kết tĩnh) hay lúc chạy chương trình (liên kết động)
- Đa hình
  - Nhìn nhận một đối tượng dưới nhiều kiểu khác nhau



Bài tập

Bài tập 1

- Kiểm tra các đoạn mã sau đây và vẽ sơ đồ lớp tương ứng

```
abstract public class Animal {
   abstract public void greeting();
}
public class Cat extends Animal {
   public void greeting() {
      System.out.println("Meow!");
   }
}
public class Dog extends Animal {
   public void greeting() {
      System.out.println("Woof!");
   }
   public void greeting(Dog another) {
      System.out.println("Wooooooooooof!");
   }
}
public class BigDog extends Dog {
   public void greeting() {
      System.out.println("Woow!");
   }
   public void greeting(Dog another) {
      System.out.println("Wooooooowwwww!");
   }
}
```

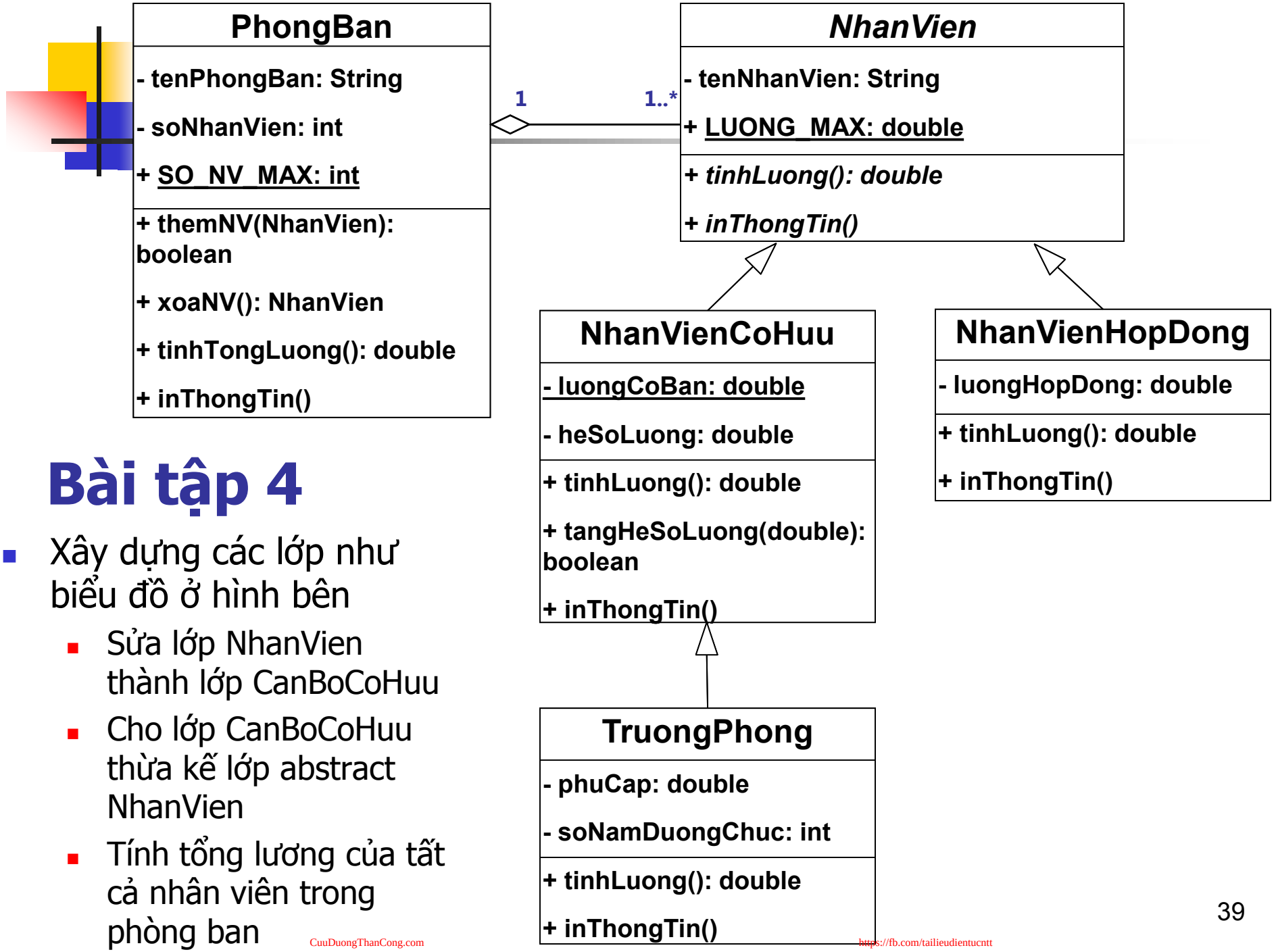
Bài tập 2

- Giải thích các đầu ra (hoặc các lỗi nếu có) cho chương trình thử nghiệm sau:

```
public class TestAnimal {
   public static void main(String[] args) {
      // Using the subclasses
      Cat cat1 = new Cat();
      cat1.greeting();
      Dog dog1 = new Dog();
      dog1.greeting();
      BigDog bigDog1 = new BigDog();
      bigDog1.greeting();

      // Using Polymorphism
      Animal animal1 = new Cat();
      animal1.greeting();
      Animal animal2 = new Dog();
      animal2.greeting();
      Animal animal3 = new BigDog();
      animal3.greeting();
      Animal animal4 = new Animal();

      // Downcast
      Dog dog2 = (Dog)animal2;
      BigDog bigDog2 = (BigDog)animal3;
      Dog dog3 = (Dog)animal3;
      Cat cat2 = (Cat)animal2;
      dog2.greeting(dog3);
      dog3.greeting(dog2);
      dog2.greeting(bigDog2);
      bigDog2.greeting(dog2);
      bigDog2.greeting(bigDog1);
   }
}
```

Bài tập 3

- Phân tích xây dựng các lớp như mô tả sau:
  - Hàng điện máy <mã hàng, tên hàng, nhà sản xuất, giá, thời gian bảo hành, điện áp, công suất>
  - Hàng sành sứ < mã hàng, tên hàng, nhà sản xuất, giá, loại nguyên liệu>
  - Hàng thực phẩm <mã hàng, tên hàng, nhà sản xuất, giá, ngày sản xuất, ngày hết hạn dùng>
- Viết chương trình tạo mỗi loại một mặt hàng cụ thể. Xuất thông tin về các mặt hàng này.



Bài tập 4

- Xây dựng các lớp như biểu đồ ở hình bên
  - Sửa lớp NhanVien thành lớp CanBoCoHuu
  - Cho lớp CanBoCoHuu thừa kế lớp abstract NhanVien
  - Tính tổng lương của tất cả nhân viên trong phòng ban