TÍNH ĐA HÌNH
POLYMORPHYSM

ĐẠI HỌC SÀI GÒN

09/2023

MỤC TIÊU

- *Khái niệm đa hình.*
- *Thể hiện Đa hình trong lập trình*

TÍNH ĐA HÌNH

Tính Đa hình

- Đa hình là khả năng một đối tượng có thể nhận nhiều hình thức.
- Trong Java, đa hình cho phép sử dụng một tham chiếu của lớp cha để tham chiếu đến một đối tượng của lớp con.

Làm thế nào thể hiện Đa hình trong lập trình?

- Đa hình thông qua Phương thức
- Đa hình thông qua Class

Đa hình thông qua Phương thức

- Đa hình tại biên dịch (Compile-time Polymorphism)
- Cùng tên phương thức nhưng số lượng hoặc kiểu dữ liệu của tham số khác nhau.
- **method overloading**
- Ví dụ
  - void sum(int a, int b) { ... }
  - void sum(double a, double b) { ... }

Đa hình thông qua Phương thức

- Đa hình tại thời gian chạy (Run-time Polymorphism)
- Phương thức của lớp cha được ghi đè bởi phương thức của lớp con.
- Sử dụng từ khóa @Override để chỉ ra rằng một phương thức ghi đè phương thức của lớp cha.

```
@Override
public void sleep() {
    System.out.println("Animal.Dog.sleep()");
}
```

Đa hình thông qua Kiểu Class

- Class-based Polymorphism
- Một đối tượng của lớp con có thể được xem như một đối tượng của lớp cha.
- Khả năng này cho phép sử dụng một tham chiếu của lớp cha để tham chiếu đến một đối tượng của lớp con.

Ví dụ

```java
public class Vehicle {
    void move() {
        System.out.println("Xe đang chạy");
    }
}

public class Car extends Vehicle {
    @Override
    void move() {
        System.out.println("Xe hơi đang chạy");
    }
}
```

```java
public class TestPolymorphism {
    public static void main(String[] args) {
        Vehicle myCar = new Car();
        myCar.move();
    }
}
```

Ích lợi của Đa hình thông qua Kiểu Class

- Tính linh hoạt: Cho phép viết mã có thể xử lý nhiều kiểu đối tượng khác nhau một cách dễ dàng.
- Mở rộng dễ dàng: Có thể thêm các lớp con mà không cần sửa đổi nhiều ở mã hiện có.
- Sử dụng chung: Có thể sử dụng một phương thức đã được định nghĩa ở lớp cha mà không cần viết lại.

Nhắc lại Casting

```
public class Vehicle {
    void move() {
        System.out.println("Xe di chuyển");
    }
}

public class Car extends Vehicle {
    void stop() {
        System.out.println("Xe hơi dừng");
    }
}
```

Nhắc lại Casting

```
public class TestCasting {
    public static void main(String[] args) {
        Vehicle myVehicle = new Car();  // Upcasting
        myVehicle.move();

        // Downcasting
        Car myCar = (Car) myVehicle;
        myCar.honk();
    }
}
```

Nhắc lại Casting

- **Upcasting**: Khi upcast một đối tượng từ lớp con lên lớp cha (như Car lên Vehicle), đang giảm chi tiết, nghĩa là đang giới hạn chỉ sử dụng những phương thức và thuộc tính được định nghĩa trong lớp cha.
- Bất cứ thứ gì đặc biệt (phương thức hoặc thuộc tính) của riêng lớp con sẽ không thể truy cập.

⇨ ***Upcasting*** *an toàn vì lớp con luôn là một thể hiện của lớp cha.*

Nhắc lại Casting

- **Downcasting**: Khi downcast một đối tượng từ lớp cha xuống lớp con (như Vehicle xuống Car), đang cố gắng lấy lại những chi tiết đó.
- Điều này chỉ an toàn nếu đối tượng thực sự là một thể hiện của lớp con đang cố gắng ép kiểu.
- Trường hợp, myVehicle thực sự chỉ là một thể hiện của Vehicle, không phải Car. Khi downcast nó thành Car, JVM sẽ ném ra một ClassCastException khi cố gắng thực hiện hành động này.

Instanceof

- Sử dụng instanceof giúp tránh gặp phải ClassCastException bằng cách kiểm tra trước khi thực hiện downcasting.

```
if (myVehicle instanceof Car) {
    Car myCar = (Car) myVehicle;
} else {
    System.out.println("Không thể down Casting về Car");
}
```

Khi nào dùng downCasting

- Downcasting được sử dụng khi muốn truy cập đến các tính năng đặc biệt (thuộc tính hoặc phương thức) của lớp con mà lớp cha không có.
- Xử lý đối tượng trong các Collection đa dạng: các đối tượng thuộc các lớp khác nhau nhưng đều kế thừa từ một lớp cha chung, muốn xử lý mỗi đối tượng dựa trên loại thực sự của nó.

Khi nào dùng downCasting

- Ví dụ: danh sách các đối tượng Animal và muốn kiểm tra nếu một đối tượng thực sự là kiểu Dog, gọi phương thức bark() chỉ dành riêng cho Dog.

```
public void diemDanh(){
    for (Animal animal : listAnimal) {
        if (animal instanceof Dog) {
            Dog dog = (Dog) animal;
                dog.bark();
        }
    }
}
```

Khi nào dùng downCasting

- Khi sử dụng các API hoặc thư viện bên ngoài mà trả về một đối tượng của lớp cha, nhưng biết chắc chắn rằng nó thực sự là một thể hiện của lớp con và muốn sử dụng các tính năng đặc biệt của lớp con đó.

Tính đa hình và từ khóa final

- Nếu một phương thức được đánh dấu là final, thì nó không thể được ghi đè.
- Từ khóa final giúp ngăn chặn việc ghi đè không mong muốn.

TRAO ĐỔI


SGU
TRƯỜNG ĐẠI HỌC SÀI GÒN
SAIGON UNIVERSITY
ĐẠI HỌC SÀI GÒN
CẤM ĐẬU XE