HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
Posts and Telecommunications Institute of Technology

RIPT

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Nguyễn Việt Dũng

dungnv1@ptit.edu.vn

TÍNH KẾ THỪA

Kết hợp (Aggregation)

"has-a"

Một đối tượng chứa (hoặc sở hữu) đối tượng khác nhưng vẫn cho phép đối tượng được chứa sống độc lập. Biểu diễn sự kết hợp giữa hai lớp mà một lớp có thể là thành viên của lớp khác, nhưng vòng đời của hai đối tượng đó là độc lập với nhau

Ví dụ:

- Sinh viên – Lớp
- Nhân viên – Công ty

Kết hợp chặt chẽ (Composition)

“contains-a"

Quan hệ mạnh mẽ hơn **Aggregation**, trong đó một đối tượng chứa đối tượng khác, và vòng đời của đối tượng con phụ thuộc hoàn toàn vào đối tượng cha. Nếu đối tượng cha bị hủy, thì đối tượng con cũng sẽ bị hủy theo.

Ví dụ:

- Phòng – Ngôi nhà
- Chương – Quyển sách

Bài toán

Xây dựng hệ thống quản lý thông tin về nhân sự trong một trường đại học. Trong hệ thống này, có nhiều loại nhân viên khác nhau như giảng viên, cán bộ hành chính. Mỗi loại nhân viên có thông tin cơ bản như tên, tuổi, và mức lương, nhưng mỗi loại lại có cách tính lương khác nhau dựa trên quy định của từng vị trí.

Tính kế thừa

Tính kế thừa cho phép xây dựng một lớp mới (lớp Con/lớp dẫn xuất), kế thừa và tái sử dụng các thuộc tính, phương thức dựa trên lớp cũ (lớp Cha) đã có trước đó.

Tạo ra lớp mới bằng cách mở rộng chức năng của một lớp cũ đã tồn tại

Tính kế thừa

- Lớp dẫn xuất là một loại của lớp cha.
- Lớp dẫn xuất có thể sử dụng các thuộc tính và phương thức của lớp cha tương tự như sử dụng các thuộc tính và phương thức của mình.
- Lớp dẫn xuất có thể mở rộng chức năng bằng cách khai báo thêm thuộc tính, phương thức đặc trưng của lớp hoặc ghi đè (override) phương thức từ lớp cha.

super

Dùng để tương tác với lớp cha từ lớp cơ sở:

– Gọi hàm constructor của lớp cha với tham số: **super(parameter list);**

– Truy vấn thuộc tính, phương thức của lớp cha: **super.variable**; **super.method(parameter list);**

Override

Cho phép các lớp con (subclass) cung cấp cách triển khai cụ thể của một phương thức đã được định nghĩa trong lớp cha (superclass). Khi sử dụng @Override, phương thức trong lớp con sẽ thay thế (override) phương thức có cùng tên và tham số của lớp cha.

```
// Ghi đè phương thức tính lương
@Override
public double calculateSalary() {
    return salary + managementAllowance;
}
```

Tại sao cần Override?

- **Đa hình (Polymorphism)**: Cho phép các đối tượng có kiểu lớp cha nhưng thực hiện các hành vi khác nhau dựa trên lớp con của chúng.
- **Tùy chỉnh hành vi**: Mỗi lớp con có thể cung cấp cách triển khai riêng cho các phương thức mà chúng kế thừa, phù hợp với yêu cầu của chúng.
- **Kiểm tra lỗi**: Sử dụng @Override giúp đảm bảo rằng bạn đang ghi đè chính xác phương thức từ lớp cha. Nếu phương thức bị viết sai (ví dụ: tên hoặc tham số), trình biên dịch sẽ báo lỗi.

final

- Nếu khai báo cho một lớp thì có thể cho phép lớp đó không thể có lớp con kế thừa
- Nếu khai báo với phương thức thì phương thức đó không thể bị ghi đè (override)
- Nếu khai báo với một biến thì biến đó không thể thay đổi giá trị sau khi khởi tạo

Khai báo

```
[access-modifier] class Subclass-name extends Superclass-name {
    // New added variables

    // New methods

    // Overridden methods
}
```

Kế thừa đơn (Single Inheritance)

Một lớp con kế thừa từ **một lớp cha** duy nhất.

Lớp con có thể truy cập và sử dụng tất cả các thuộc tính và phương thức (public, protected) của lớp cha.

Chỉ có thể có **một lớp cha** cho một lớp con.

Cú pháp:

```
class Parent {
// Thuộc tính và phương thức của lớp cha
}
class Child extends Parent {
// Thuộc tính và phương thức của lớp con
}
```

Kế thừa bội (Mutiple Inheritance)

Java **không hỗ trợ kế thừa bội** trực tiếp giữa các lớp vì có thể xảy ra xung đột trong việc chọn phương thức từ các lớp cha. Tuy nhiên, Java hỗ trợ kế thừa bội thông qua việc sử dụng giao tiếp (**interfaces)**. Một lớp có thể cài đặt nhiều giao tiếp. Các giao tiếp được phân cách nhau bởi dấu phẩy. Khi đó, lớp phải cài đặt cụ thể tất cả các phương thức của tất cả các giao tiếp mà nó sử dụng.

Qua đó:

- Tăng tính linh hoạt: Một lớp có thể thực thi nhiều hành vi mà không bị ràng buộc với một cấu trúc kế thừa duy nhất
- Giảm sự phức tạp và tránh xung đột phương thức (Diamond Problem).

Các giao tiếp:

Kế thừa bội (Mutiple Inheritance)

Bài toán:

Phát triển một ứng dụng quản lý các thiết bị điện tử. Trong đó, có một số thiết bị có thể thực hiện nhiều vai trò như **phát nhạc**, **chụp ảnh**, và thậm chí **gọi điện**. Cụ thể, hệ thống bao gồm 3 loại thiết bị:

- **MusicPlayer**: Phát nhạc.
- **Camera**: Chụp ảnh.
- **Smartphone:** Phát nhạc, Chụp ảnh

```
// Interface mô tả hành vi phát nhạc
interface MusicPlayer {
    void playMusic();
}

// Interface mô tả hành vi chụp ảnh
interface Camera {
    void takePhoto();
}

// Lớp Smartphone thực thi cả hai interface
class Smartphone implements MusicPlayer, Camera {
    @Override
    public void playMusic() {
        System.out.println("Playing music on Smartphone.");
    }

    @Override
    public void takePhoto() {
        System.out.println("Taking a photo with Smartphone.");
    }
}
```

Bài tập 1: Kế thừa đơn - Quản lý nhân viên

Phát biểu bài toán: Xây dựng hệ thống quản lý nhân viên cho một công ty. Mỗi nhân viên có **tên**, **tuổi**, **tiền lương**. Ngoài ra, có hai loại nhân viên với cách tính lương khác nhau:

1. Nhân viên văn phòng (OfficeEmployee): Lương của nhân viên văn phòng được tính dựa trên **số ngày làm việc** trong tháng. **Mức lương mỗi ngày** là một hằng số cố định với mọi nhân viên (ví dụ: 100 đơn vị tiền/ngày).

Công thức tính lương nhân viên văn phòng:

$$\text{Lương} = \text{Số ngày làm việc} \times \text{Lương mỗi ngày}$$

2. Nhân viên kỹ thuật (TechnicalEmployee): Lương của nhân viên kỹ thuật được tính dựa trên **số giờ làm việc** trong tháng và **tiền công theo giờ**. Mỗi nhân viên có mức tiền công theo giờ khác nhau.

Công thức tính lương nhân viên kỹ thuật:

$$\text{Lương} = \text{Số giờ làm việc} \times \text{Tiền công theo giờ}$$

Bài tập 2: Xây dựng một hệ thống quản lý nhân viên trong một công ty có nhiều loại nhân viên với các chức năng khác nhau. Các loại nhân viên bao gồm:

- Nhân viên văn phòng (OfficeEmployee): Có thể thực hiện chức năng gửi email.
- Nhân viên kỹ thuật (TechnicalEmployee): Có thể thực hiện chức năng lập trình.
- Nhân viên bán hàng (SalesEmployee): Có thể thực hiện chức năng bán hàng.

Yêu cầu: Sử dụng kế thừa bội qua **interfaces** để mô tả các hành vi của nhân viên. Tạo các interfaces:

- EmailSender: mô tả chức năng gửi email.
- Programmer: mô tả chức năng lập trình.
- Salesperson: mô tả chức năng bán hàng.

Tạo các lớp OfficeEmployee, TechnicalEmployee, và SalesEmployee kế thừa bội từ các interfaces trên. Viết chương trình cho phép sử dụng các chức năng của từng loại nhân viên.