Tính kế thừa trong Java

Tính kế thừa trong Java là môt kỹ thuật mà trong đó một đối tượng thu được tất cả thuộc tính và hành vi của đối tượng cha. Ý tưởng đằng sau tính kế thừa trong Java là bạn có thể tạo các lớp mới mà được xây dựng dựa trên các lớp đang tồn tại. Khi bạn kế thừa từ một lớp đang tồn tại, bạn có thể tái sử dụng các phương thức và các trường của lớp cha, và bạn cũng có thể bổ sung thêm các phương thức và các trường khác. Tính kế thừa biểu diễn mối quan hệ IS-A, còn được gọi là mối quan hệ cha-con.

Khi chúng ta nói về tính kế thừa, từ khóa thường xuyên nhất được sử dụng là **extends** và**implements**. Những từ khóa này có thể định nghĩa một kiểu là loại IS-A của loại khác. Sử dụng những từ khóa, chúng ta có thể tạo một đối tượng sử dụng thuộc tính của đối tượng khác. Chúng ta sử dụng từ khóa extends của lớp con để có thể kế thừa các thuộc tính của lớp cha trừ các thuộc tính private của lớp cha.

Tại sao sử dụng tính kế thừa trong Java?

- Để ghi đè phương thức (Method Overriding), do đó có thể thu được tính đa hình tại runtime.
- Để làm tăng tính tái sử dụng của code.

Cú pháp của Tính kế thừa trong Java

```
class ten_lop_con extends ten_lop_cha
{
   //cac phuong thuc va cac truong
}
```

Từ khóa extends chỉ rằng bạn đang tạo một lớp mới mà kế thừa từ một lớp đang tồn tại. Trong Java, một lớp mà được kế thừa được gọi là một lớp cha. Lớp mới được gọi là lớp con.

Trong ví dụ sau, Programmer là lớp con và Employee là lớp cha. Mối quan hệ giữa hai lớp là Programmer IS-A Employee. Nghĩa là Programmer là một kiểu của Employee.

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
```

```
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Luong Lap trinh vien la:"+p.salary);
   System.out.println("Bonus cua Lap trinh vien la:"+p.bonus);
}
}
```

Ở trên, đối tượng Programmer có thể truy cập trường của riêng lớp nó cũng như của lớp Employee, đó là ví dụ cho tính tái sử dụng.

Các loại kế thừa trong Java

Trên cơ sở các lớp thì có 3 loại kế thừa trong Java, đó là *single (đơn), multilevel (nhiều tầng) và hierarchical (có cấu trúc)*. Trong lập trình Java, *đa kế thừa (multiple) và kế thừa lai (hybrid)*chỉ được hỗ trợ thông qua Interface. Chúng ta sẽ tìm hiểu về Interface trong chương sau đó.

Ghi chú: Đa kế thừa không được hỗ trợ trong Java thông qua lớp. Khi một lớp kế thừa từ nhiều lớp, thì đây là đa kế thừa.

Câu hỏi: Tại sao đa kế thừa không được hỗ trợ trong Java thông qua lớp?

Trả lời: Để giảm tính phức tạp và làm đơn giản hóa ngôn ngữ, đa kế thừa không được hỗ trợ trong Java. Giả sử có tình huống có ba lớp là A, B và C. Lớp C kế thừa lớp A và B. Nếu các lớp A và B có cùng phương thức và bạn gọi nó từ đối tượng lớp con, thì điều này gây là tính lưỡng nghĩa là để gọi phương thức của lớp A hoặc lớp B.

Bởi vì, compile time error thì tốt hơn là runtime error, Java sẽ thông báo một compile time error nếu bạn kế thừa 2 lớp. Do đó, dù bạn có hay không có cùng phương thức hay khác phương thức, thì đó cũng là một lỗi tại compile time.

```
class A{
void msg(){System.out.println("Hello");}
}
class B{
void msg(){System.out.println("Welcome");}
}
class C extends A,B
{ //gia su neu no da co
```

```
 Public Static void main(String args[]){
   C obj=new C();
   obj.msg();//Bay gio phuong thuc msg() nao se duoc goi?
}
}
```

Chương trình trên sẽ cho một Compile Time Error.

Khi bạn đã hiểu rõ về từ khóa **extends**, chúng ta cùng tìm hiểu về từ khóa **implements**trong quan hệ IS-A.

Từ khóa **implements** được sử dụng bởi các lớp mà kế thừa từ Interface. Interface có thể không bao giờ được kết thừa bởi các lớp.

Ví dụ:

```
public interface A {}

public class B implements A{
}

public class C extends B{
}
```

Từ khóa instanceof trong Java

Bây giờ chúng ta sẽ dùng toán tử **instanceof** để kiểm tra xem B có phải là một A và dog có phải là một A.

```
interface A{}

class B implements A{}

public class C extends B{
   public static void main(String args[]){
```

```
        B m = new B();
        C d = new C();

        System.out.println(m instanceof A);
        System.out.println(d instanceof B);
        System.out.println(d instanceof A);
    }
}
```

Kết quả in ra sẽ như sau:

```
true
true
true
```

Quan hệ HAS-A trong Java

Có những quan hệ chủ yếu dựa vào cách sử dụng. Nó xác định có hay không một lớp cụ thể HAS-A. Quan hệ này giúp chúng ta giảm được dư thừa trong code cũng như tránh các bug.

Cùng xem ví dụ dưới đây:

```
public class Vehicle{}
public class Speed{}
public class Van extends Vehicle{
          private Speed sp;
}
```

Điều này chỉ ra rằng lớp Van có quan hệ HAS-A với lớp Speed. Việc sử dụng lớp riêng rẽ cho lớp Speed, chúng ta không cần thiết phải đặt toàn bộ code của lớp Speed bên trong lớp Van, điều này tăng tính tái sử dụng của lớp Speed cho nhiều ứng dụng.

Một đặc điểm quan trọng nữa phải ghi nhớ là Java chỉ hỗ trợ kế thừa đơn. Điều này nghĩa là một lớp không thể kế thừa từ nhiều hơn một lớp. Do đó, đoạn code dưới đây là không hợp lệ:

```
public class extends A, B{}
```

Mặc dù vậy một lớp vẫn có thể implement một hoặc nhiều interface. Điều này loại bỏ khả năng không thể đa kế thừa trong Java.