Inheritance - Kế thừa

Lớp cha - Lớp con

Kế thừa là cơ chế cho phép lớp CON sử dụng lại các thuộc tính và hành vi đã được định nghĩa trong lớp CHA.

- Quan hệ giữa lớp con và lớp cha là quan hệ "is-a" (cái này là 1 cái kia).

- Lớp được kế thừa: lớp cha (parent class), lớp cơ sở (base class) hoặc super class.

- Lớp kế thừa: lớp con (child class), lớp dẫn xuất (derived class) hoặc sub class.

- Lớp con kế thừa tất cả các thành phần lớp cha ngoại trừ các thành phần được khai báo là "private".

- Contructor không được kế thừa.

- Lớp con có thế gọi contructor của lớp cha.

- Lớp con phải tự định nghĩa Contructor riêng.

- Lớp con có thể định nghĩa thêm các thuộc tính và phương thức mới

- Java KHÔNG cho phép đa kế thừa (1 lớp kế thừa nhiều lớp cha).

- Các loại kế thừa:

+ Kế thừa đơn (Single Inheritance): Kế thừa đơn trong Java là quá trình kế thừa các thuộc tính và phương thức của một lớp cha đơn lẻ. Trong kế thừa đơn, một lớp con chỉ được kế thừa từ một lớp cha duy nhất.

Ví dụ:

class Cha {

// thuộc tính và phương thức của lớp Cha

}

class Con extends Cha {

// thuộc tính và phương thức của lớp Con

}

+ Kế thừa đa hình (Multilevel Inheritance): Kế thừa đa hình trong Java là quá trình kế thừa từ nhiều lớp cha để tạo ra một lớp con mới. Trong kế thừa đa hình, một lớp con có thể kế thừa từ một lớp cha và đồng thời là lớp cha của một lớp con khác.

Ví dụ:

class A {

// thuộc tính và phương thức của lớp A

}

class B extends A {

// thuộc tính và phương thức của lớp B

}

class C extends B {

// thuộc tính và phương thức của lớp C

}

- Cú pháp:

Class tên lớp con extends tên lớp cha

VD:

Public class Person{

// các thuộc tính và phương thức của lớp Person

}

public class Employee extends Person {

// các thuộc tính và phương thức của lớp Employee

}

Lớp con thường chứa nhiều thông tin hơn lớp cha của nó

- Private ở lớp cha không thể được truy cập từ lớp con nhưng lớp

con có thể truy cập thông qua các phương thức mà lớp cha đã public

- Lợi ích của quan hệ thừa kế:

+ Tránh lặp các đoạn mã bị trùng lặp.

+ Định nghĩa được một giao thức chung cho tập các lớp gắn kết với nhau bởi quan hệ thừa kế.

Ghi đè phương thức - Method overriding

Từ khoá: super

Đại diện cho lớp cha và có thể sử dụng để gọi:

+ Phương thức ở lớp cha.

+ Contructor ở lớp cha.

- Cú pháp:

super(): gọi tới contructor không tham số;

hoặc

super (parameters): gọi tới contructor có tham số

- Cả 2 lệnh trên phải là dòng lệnh đầu tiên trong contructor ==> cách duy nhất để gọi tường minh contructor của lớp cha

- Gọi tới phương thức của lớp cha: super.method();

Ghi đè phương thức - Method overriding

Từ khoá: @Override

- Là cơ chế cho phép lớp con định nghĩa lại các phương thức đã được định nghĩa trước đó ở lớp cha.

- Phương thức override ở lớp con phải:

+ Cùng tên

+ Cùng danh sách tham số

+ Cùng kiểu dữ liệu trả về với phương thức cha

+ Phải có access modifier có level >= so với phương thức ở lớp cha.

- Một phương thức của đối tượng chỉ có thể được ghi đè nếu nó có thể truy cập được.

==> phương thức private ở lớp cha không thể bị ghi đè.

- Phương thức tĩnh có thể được kế thừa nhưng không thể được ghi đè

==> Nếu định nghĩa lại phương thức tĩnh trong lớp con ==> phương thức ở lớp cha sẽ bị ẩn

==> Có thể gọi phương thức bị ẩn này = cú pháp SuperCLassName.staticMethodName.

Ghi đè (Overriding) và Nạp chồng (Overloading)

- Overriding:

+ Phương thức đã được đĩnh nghĩa ở lớp cha, phương thức ở lớp con phải cùng (tên, bộ tham số, kiểu trả về) với phương thức ở lớp cha.

+ Xảy ra trong 2 class có quan hệ kế thừa.

=> Ví dụ về đa hình lúc runtime.

Giả sử có một lớp cha Animal và hai lớp con là Cat và Dog, tất cả đều có một phương thức makeSound(). Ví dụ:

class Animal {

public void makeSound() {

System.out.println("The animal makes a sound");

}

}

class Cat extends Animal {

public void makeSound() {

System.out.println("Meow");

}

}

class Dog extends Animal {

public void makeSound() {

System.out.println("Woof");

}

}

Giờ đây, trong phương thức main(), chúng ta có thể tạo một mảng động chứa các đối tượng Animal và gọi phương thức makeSound() trên mỗi đối tượng. Ví dụ:

public static void main(String[] args) {

Animal[] animals = new Animal[2];

animals[0] = new Cat();

animals[1] = new Dog();

for (Animal animal : animals) {

animal.makeSound();

}

}

Kết quả khi chạy chương trình sẽ là:

Meow

Woof

- Overloading:

+ Định nghĩa ra nhiều phương thức với cùng tên nhưng không có cùng tham số.

+ Được thực hiện bên trong 1 class.

=> Ví dụ về đa hình lúc biên dịch.

Giả sử có một lớp cha Shape và hai lớp con là Circle và Rectangle, tất cả đều có một phương thức draw(). Ví dụ:

class Shape {

public void draw() {

System.out.println("Drawing a shape");

}

}

class Circle extends Shape {

public void draw() {

System.out.println("Drawing a circle");

}

}

class Rectangle extends Shape {

public void draw() {

System.out.println("Drawing a rectangle");

}

}

Giờ đây, trong phương thức main(), chúng ta có thể tạo một đối tượng Shape và gọi phương thức draw() trên đối tượng đó. Ví dụ:

public static void main(String[] args) {

Shape shape = new Circle();

shape.draw();

}

Ở đây, chúng ta tạo một đối tượng Circle và gán nó cho một biến Shape. Tuy nhiên, vì Circle là một lớp con của Shape và ghi đè phương thức draw(), phương thức draw() được gọi sẽ là phương thức draw() của lớp Circle.

Khi chạy chương trình, kết quả sẽ là:

Drawing a circle

Lớp Object

- Là lớp gốc của các lớp Java.

- Nếu không có kế thừa nào được chỉ định khi một lớp được định nghĩa, thì lớp cha mặc định của lớp đó là Object.

- Tất cả các lớp đều kế thừa từ lớp Object.

toString()

- Trả về 1 chuỗi mô tả đối tuợng.

- Có thể dùng System.out.println(object) để thay thế cho System.out.println(object.toString ())

Đa hình là khả năng 1 đối tượng có thể tiếp nhận nhiều hình dạng khác nhau.