第01天 java面向对象

今日内容介绍

* 继承
* 抽象类

# 继承

## 继承的概述

在现实生活中，继承一般指的是子女继承父辈的财产。在程序中，继承描述的是事物之间的所属关系，通过继承可以使多种事物之间形成一种关系体系。

## 继承的格式&使用

在程序中，如果想声明一个类继承另一个类，需要使用extends关键字。

格式：

class 子类 extends 父类 {}

### 案例代码一:

**package** com.itheima\_01;

/\*

\* 继承：多个类有共同的成员变量和成员方法，抽取到另外一个类中（父类），在让多个类去继承这个父类，我们的多个类就可以获取到父类中的成员了。

\* extends

\*

\*/

**public** **class** ExtendsDemo {

**public** **static** **void** main(String[] args) {

DotA1 d = **new** DotA1();

d.start();

LOL1 l = **new** LOL1();

l.start();

}

}

**class** Game1 {

String name;

**double** version;//版本号

String agent;//代理商

**public** **void** start() {

System.*out*.println("游戏启动了");

}

**public** **void** stop() {

System.*out*.println("游戏关闭了");

}

}

**class** DotA1 **extends** Game1 {

/\*String name;

double version;//版本号

String agent;//代理商

public void start() {

System.out.println("游戏启动了");

}

public void stop() {

System.out.println("游戏关闭了");

}\*/

}

**class** LOL1 **extends** Game1 {

/\*String name;

double version;//版本号

String agent;//代理商

public void start() {

System.out.println("游戏启动了");

}

public void stop() {

System.out.println("游戏关闭了");

}\*/

}

## 继承的特点

在类的继承中，需要注意一些问题，具体如下：

1、在Java中，类只支持单继承，不允许多继承，也就是说一个类只能有一个直接父类，例如下面这种情况是不合法的。

class A{}

class B{}

class C extends A,B{} // C类不可以同时继承A类和B类

2、多个类可以继承一个父类，例如下面这种情况是允许的。

class A{}

class B extends A{}

class C extends A{} // 类B和类C都可以继承类A

3、在Java中，多层继承是可以的，即一个类的父类可以再去继承另外的父类，例如C类继承自B类，而B类又可以去继承A类，这时，C类也可称作A类的子类。下面这种情况是允许的。

class A{}

class B extends A{} // 类B继承类A，类B是类A的子类

class C extends B{} // 类C继承类B，类C是类B的子类，同时也是类A的子类

4、在Java中，子类和父类是一种相对概念，也就是说一个类是某个类父类的同时，也可以是另一个类的子类。例如上面的这种情况中，B类是A类的子类，同时又是C类的父类

### 案例代码二

**package** com.itheima\_01;

/\*

\* Java中继承的特点：

\* Java语言只支持单一继承，只能继承一个父类（一个儿子只能有一个亲爹）

\* Java语言支持多层继承（一个儿子可以有一个亲爹，还可以有一个亲爷爷）

\*

\*/

**public** **class** ExtendsDemo2 {

**public** **static** **void** main(String[] args) {

LOL l = **new** LOL();

l.update();

l.start();

}

}

**class** Game {

**public** **void** start() {

System.*out*.println("游戏启动了");

}

}

**class** PCGame **extends** Game {

**public** **void** update() {

System.*out*.println("PCGame更新了");

}

}

**class** MobileGame **extends** Game {

**public** **void** update() {

System.*out*.println("MobileGame更新了");

}

}

**class** LOL **extends** PCGame {

}

## 继承中成员变量的特点

A:子类只能获取父类非私有成员

子父类中成员变量的名字不一样直接获取父类的成员变量

子父类中成员变量名字是一样的获取的是子类的成员变量

B:就近原则：谁离我近我 就用谁

如果有局部变量就使用局部变量

如果没有局部变量，有子类的成员变量就使用子类的成员变量

如果没有局部变量和子类的成员变量，有父类的成员变量就使用父类的成员变量

C: super:可以获取父类的成员变量和成员方法,用法和this是相似的

### 案例代码三

**package** com.itheima\_01;

/\*

\* 继承中成员变量的特点

\* 子类只能获取父类非私有成员

\* 子父类中成员变量的名字不一样直接获取父类的成员变量

\* 子父类中成员变量名字是一样的获取的是子类的成员变量

\*

\* 就近原则：谁离我近我就用谁

\* 如果有局部变量就使用局部变量

\* 如果没有局部变量，有子类的成员变量就使用子类的成员变量

\* 如果没有局部变量和子类的成员变量，有父类的成员变量就使用父类的成员变量

\* 啥都没有，出错了！！！

\*

\* super:可以获取父类的成员变量和成员方法,用法和this是相似的

\*/

**public** **class** ExtendsDemo3 {

**public** **static** **void** main(String[] args) {

Kid3 k = **new** Kid3();

k.show();

}

}

**class** Dad3 {

String name = "建霖";

}

**class** Kid3 **extends** Dad3 {

String name = "四葱";

**public** **void** show() {

String name = "五葱";

System.*out*.println(**super**.name);

System.*out*.println(**this**.name);

System.*out*.println(name);

}

}

## 继承中成员方法的特点&方法重写

### 案例代码四

A:子类中没有这个方法,调用父类的

**package** com.itheima\_01;

/\*

\* 继承中成员方法的特点

\* 子类中没有这个方法，调用父类的

\*/

**public** **class** ExtendsDemo4 {

**public** **static** **void** main(String[] args) {

Kid4 k = **new** Kid4();

k.eat();

}

}

**class** Dad4 {

**public** **void** eat() {

System.*out*.println("小酌两口");

System.*out*.println("去睡觉了");

}

}

**class** Kid4 **extends** Dad4 {

}

### 案例代码五

B: 子类中重写了这个方法，调用子类的

方法的重写：在子父类当中，子类的方法和父类的完全一样，子类重写了父类的方法（覆盖），当子类重写了父类的方法之后，使用子类对象调用的就是子类的方法

**package** com.itheima\_01;

/\*

\* 继承中成员方法的特点

\* 子类中没有这个方法，调用父类的

\* 子类中重写了这个方法，调用子类的

\*

方法的重写：在子父类当中，子类的方法和父类的完全一样，子类重写了父类的方法（覆盖），当子类重写了父类的方法之后，使用子类对象调用的就是子类的方法

方法的重载：在一个类中，有多个重名的方法，但是其参数不一样（参数的个数，参数的类型，参数的顺序），和返回值无关

\*/

**public** **class** ExtendsDemo4 {

**public** **static** **void** main(String[] args) {

Kid4 k = **new** Kid4();

k.eat();

}

}

**class** Dad4 {

**public** **void** eat() {

System.*out*.println("小酌两口");

System.*out*.println("去睡觉了");

}

}

**class** Kid4 **extends** Dad4 {

**public** **void** eat() {

System.*out*.println("好好吃饭");

}

}

## 方法重写的应用场景&注意事项

方法重写的应用场景：当父类的方法不能完全满足子类使用的时候，既可以保留父类的功能（沿袭、传承），还可以有自己特有的功能

方法重写的注意事项：

不可以重写父类私有的成员方法，压根就看不到父类的私有成员

子类重写父类方法，权限必须大于等于父类方法的权限

注解

@Override：方法重写，说明下面的方法是重写父类的方法

#### 案例代码六

package com.itheima\_03;

/\*

\* 方法重写的应用场景：当父类的方法不能完全满足子类使用，这个时候子类重写父类的方法，

\* 并可以在方法中使用关键字super调用父类的方法，这样做即可以保有父类的功能，也可以拥有子类特有的功能

\* 方法重写的注意事项：

\* 不能重写父类私有的方法

\* 权限必须大于等于父类方法的权限

\*

\* 注解：@

\*

\*/

public class ExtendsDemo5 {

public static void main(String[] args) {

NewPhone np = new NewPhone();

np.call();

}

}

class Phone {

void call() {

System.out.println("打电话");

}

}

class NewPhone extends Phone {

@Override

public void call() {

System.out.println("录音");

//System.out.println("打电话");

//super.call();

}

}

## 继承中构造方法的执行顺序

A:super(实参列表);语句 在子类的构造方法中使用,用来调用父类中的构造方法(具体哪一个由传递的参数决定),并且只能在构造方法第一行使用

B:this(实参列表); 语句 在类的构造方法中使用,用来调用本类中的其它构造方法(具体哪一个由传递的参数决定),并且只能在构造方法的第一行使用

### 案例代码七

**package** com.itheima\_01;

/\*

\* 继承中构造方法的执行顺序

\* 在子父类中，创建子类对象，调用子类的构造方法，

\* 在子类的构造方法的第一行代码如果没有调用父类的构造或者没有调用子类的其他构造，则默认调用父类无参构造

\* 为什么要调用父类构造？

\* 因为需要给父类的成员变量初始化

\* 肯定会先把父类的构造执行完毕，在去执行子类构造中的其他代码

\*

\* 我是父类无参构造 --- 我是子类有参构造 --- 我是子类无参构造

\*/

**public** **class** ExtendsDemo6 {

**public** **static** **void** main(String[] args) {

//Die d = new Die();

Zi6 z = **new** Zi6();

}

}

**class** Die6 {

**public** Die6() {

System.*out*.println("我是父类无参构造");

}

**public** Die6(**int** num) {

System.*out*.println("我是父类有参构造");

}

}

**class** Zi6 **extends** Die6 {

**public** Zi6() {

//super(1);

//super();

**this**(1);//不会再调用父类的无参构造了

System.*out*.println("我是子类无参构造");

}

**public** Zi6(**int** num) {

//会默认调用父类无参构造

System.*out*.println("我是子类有参构造");

}

}

## this与super区别

### 案例代码八:

**package** com.itheima\_01;

/\*

\* this和super的区别

this:当前对象的引用

调用子类的成员变量

调用子类的成员方法

在子类的构造方法第一行调用子类其他构造方法

super:子类对象的父类引用

调用父类的成员变量

调用父类的成员方法

在子类的构造方法第一行调用父类的构造方法

\*/

**public** **class** ExtendsDemo7 {

**public** **static** **void** main(String[] args) {

Zi z = **new** Zi();

z.function();

}

}

**class** Die {

**int** num = 10;

**public** Die() {

System.*out*.println("我是父类无参构造");

}

**public** Die(**int** num) {

System.*out*.println("我是父类有参构造");

}

**public** **void** method() {

System.*out*.println("我是父类的方法");

}

}

**class** Zi **extends** Die {

//int num = 30;

**public** Zi() {

//this(1);//第一行不调用子类其他构造或者是父类构造，默认调用父类无参构造

**super**();

System.*out*.println("我是子类无参构造");

}

**public** Zi(**int** num) {

System.*out*.println("我是子类有参构造");

}

**public** **void** method() {

System.*out*.println("我是子类的方法");

}

**public** **void** function() {

//this.num = 50;

//System.out.println(num);

//this.method();

//super.num = 40;

//super.method();

System.*out*.println(**this**.num);

}

}

## 继承优缺点

A:优点

提高了代码的复用性

提高了代码的可维护性

B:缺点：

类的耦合性增强了

开发的原则：高内聚低耦合

内聚：就是自己完成某件事情的能力

耦合：类与类的关系

# 抽象类

## 抽象类概述

当编写一个类时，我们往往会为该类定义一些方法，这些方法是用来描述该类的功能具体实现方式，那么这些方法都有具体的方法体。

但是有的时候，某个父类只是知道子类应该包含怎么样的方法，但是无法准确知道子类如何实现这些方法。比如一个图形类应该有一个求周长的方法，但是不同的图形求周长的算法不一样。那该怎么办呢？

分析事物时，发现了共性内容，就出现向上抽取。会有这样一种特殊情况，就是方法功能声明相同，但方法功能主体不同。那么这时也可以抽取，但只抽取方法声明，不抽取方法主体。那么此方法就是一个抽象方法。

### 案例代码九:

**package** com.itheima\_01;

/\*

\* abstract:关键字，用于修饰方法和类

\* 抽象方法：不同类的方法是相似，但是具体内容又不太一样，所以我们只能抽取他的声明，没有具体的方法体，没有具体方法体的方法就是抽象方法

\* 抽象类：有抽象方法的类必须是抽象类

\*

\* 注意：一个类继承了抽象类需要重写他所有的抽象方法,否则这个类就得是抽象类

\*/

**public** **class** AbstractDemo {

}

**abstract** **class** Animal1 {

**public** **abstract** **void** eat();

//非抽象方法子类可以不重写

**public** **void** run() {

}

}

**class** Cat1 **extends** Animal1 {

@Override

**public** **void** eat() {

System.*out*.println("猫吃鱼");

}

/\*public void eat() {

System.out.println("猫吃鱼");

}\*/

}

**abstract** **class** Dog1 **extends** Animal1 {

/\*public void eat() {

System.out.println("狗吃屎");

}\*/

}

## 抽象类的特点

抽象类的特点：

抽象方法只能在抽象类里面

抽象类和抽象方法必须被abstract修饰

抽象类不能创建对象（不能实例化）

抽象类中可以有非抽象的方法

抽象类和类的关系也是继承

一个类继承了抽象类要么重写所有的抽象方法，要么他自己是抽象类

### 案例代码十

**package** com.itheima\_01;

/\*

\* 抽象类的特点：

\* 抽象方法只能在抽象类里面

\* 抽象类和抽象方法必须被abstract修饰

\* 抽象类不能创建对象（不能实例化）

\* 抽象类中可以有非抽象的方法

\* 抽象类和类的关系也是继承

\* 一个类继承了抽象类要么重写所有的抽象方法，要么他自己是抽象类

\*/

**public** **class** AbstractDemo2 {

**public** **static** **void** main(String[] args) {

//Animal a = new Animal();

}

}

**abstract** **class** Animal2 {

**public** **abstract** **void** eat();

**public** **void** run() {

}

}

**class** Cat2 **extends** Animal2 {

@Override

**public** **void** eat() {

// **TODO** Auto-generated method stub

}

}

## 抽象类的成员的特点

A:抽象类的成员特点：

成员变量

可以有成员变量

可以有常量

成员方法

可以有抽象方法

可以有非抽象方法

构造方法

可以有构造方法的，需要对抽象类的成员变量进行初始化

### 案例代码十一

**package** com.itheima\_01;

/\*

\* 抽象类的成员特点：

\* 成员变量

\* 可以有成员变量

\* 可以有常量

\* 成员方法

\* 可以有抽象方法

\* 可以有非抽象方法

\* 构造方法

\* 可以有构造方法的，需要对抽象类的成员变量进行初始化

\*

\* final:修饰类、成员变量、成员方法

\*/

**public** **class** AbstractDemo3 {

**public** **static** **void** main(String[] args) {

Dog d = **new** Dog();

d.barking();

}

}

**abstract** **class** Animal {

String name = "哮天犬";

**final** **int** num = 10;

**public** Animal() {

System.*out*.println("我是抽象类的构造方法");

}

**public** **abstract** **void** eat();

**public** **void** run() {}

}

**class** Dog **extends** Animal {

**public** **void** barking() {

System.*out*.println(name);

System.*out*.println(num);

}

@Override

**public** **void** eat() {

// **TODO** Auto-generated method stub

}

}

## 抽象类案例

### 老师案例:

老师类:

属性:姓名,年龄,性别

行为:讲课

基础班老师:

属性:姓名,年龄,性别

行为:讲基础班课程

就业班老师:

属性:姓名,年龄,性别

行为:讲就业班课程

#### 案例代码十二

**package** com.itheima\_02;

/\*

\* 基础班老湿，就业班老湿

\*

\* 共性：

\* 属性 姓名，年龄，性别

\* 行为 讲课，唱歌

\*/

**public** **class** AbstractTest {

**public** **static** **void** main(String[] args) {

BasicTeacher bt = **new** BasicTeacher();

bt.name = "风清扬";

bt.teach();

JobTeacher jt = **new** JobTeacher();

jt.name = "苍老师";

jt.teach();

}

}

**abstract** **class** Teacher {

String name;//姓名

**int** age;//年龄

String gender;//性别

//讲课

**public** **abstract** **void** teach();

}

**class** BasicTeacher **extends** Teacher {

@Override

**public** **void** teach() {

System.*out*.println(name + "讲基础班课程");

}

}

**class** JobTeacher **extends** Teacher {

@Override

**public** **void** teach() {

System.*out*.println(name + "讲就业班课程");

}

}

### 雇员案例

雇员类:

属性:姓名,编号,薪水

行为:工作

程序员:

属性:姓名,编号,薪水

行为:写代码工作

经理:

属性:姓名,编号,薪水,奖金

行为:盯着程序员写代码工作

#### 案例代码十三

**package** com.itheima\_02;

/\*

\* 雇员(Employee)示例：

需求：

公司中

程序员(programmer)有姓名(name)，工号(id)，薪水(pay)，工作内容(work)。

项目经理(Manager)除了有姓名(name)，工号(id)，薪水(pay)，还有奖金(bonus)，工作内容(work)

员工：

属性 name，id，pay

行为 work

\*/

**public** **class** AbstractTest2 {

**public** **static** **void** main(String[] args) {

Programmer p = **new** Programmer();

p.work();

Manager m = **new** Manager();

m.work();

}

}

**abstract** **class** Employee {

String name;//姓名

String id;//id

**double** pay;//薪水

//工作

**public** **abstract** **void** work();

}

**class** Programmer **extends** Employee {

@Override

**public** **void** work() {

System.*out*.println("写代码");

}

}

**class** Manager **extends** Employee {

**byte** bonus;

@Override

**public** **void** work() {

System.*out*.println("盯着程序员写代码");

}

}

### 技师案例

技师:

属性:姓名,年龄

行为:服务

足疗技师:

属性:姓名,年龄

行为:按按脚揉揉肩

其它技师:

属性:姓名,年龄

行为:你懂的

#### 案例代码十四

**package** com.itheima\_02;

/\*

\* 足疗店

\* 技师

\* 足疗技师

\* 其他技师

\*

\* 共性：

\* 属性 姓名，年龄

\* 行为 服务

\*/

**public** **class** AbstractTest3 {

**public** **static** **void** main(String[] args) {

足疗技师 zl = **new** 足疗技师();

zl.service();

其他技师 qt = **new** 其他技师();

qt.service();

}

}

**abstract** **class** 技师 {

String name;//姓名

**int** age;//年龄

//服务

**public** **abstract** **void** service();

}

**class** 足疗技师 **extends** 技师 {

@Override

**public** **void** service() {

System.*out*.println("按按脚揉揉肩");

}

}

**class** 其他技师 **extends** 技师 {

@Override

**public** **void** service() {

System.*out*.println("你懂的");

}

}

## 抽象类的细节

A:抽象类关键字abstract可以和哪些关键字共存?

1.private：

私有的方法子类是无法继承到的，也不存在覆盖，而abstract和private一起使用修饰方法，abstract既要子类去实现这个方法，而private修饰子类根本无法得到父类这个方法。互相矛盾。

2.final:

抽象类不能和final共存,因为抽象类自身无法创建对象,我们需要通过子类创建对象,一旦抽象类使用final关键字,那么抽象类就没有子类

抽象方法不能和final共存,因为抽象方法后期需要被子类重写,一旦加final无法重写

3.static:

抽象方法不能和static关键字共存,因为一旦加static我们就可以通过类名直接访问抽象方法,由于抽象方法没有方法体,没有任何意义,也不允许这样做

B:抽象类中是否可以不定义抽象方法?

是可以的，那这个抽象类的存在到底有什么意义呢？不让该类创建对象,方法可以直接让子类去使用

C:抽象类是否有构造函数?  
 有,抽象类的构造函数,是由子类的super语句来调用,用于给抽象类中的成员初始化