链滴

继承

作者：lxlcs201
原文链接：https://ld246.com/article/1533028625924
来源网站：链滴

作为面向对象的编程语言，Java也提供了类的继承机制。利用继承机制，新建的类可以建立在原有类基础之上，使用或者重写原有类的成员方法，访问原有类的成员变量。我们称新类为原有类子类，而有类为新类的父类（superclass）。如果类A是类B的父类，而类B是类C的父类，我们也称C是A的子，类C是从类A继承而来的。在Java中，类的继承是单一继承，也就是说，一个子类只能拥有一个父，一个父类可以有多个子类。所有Java的类均是由java.lang.Object类继承而来的，所以Object是所类的祖先类，而除了Object外，所有类必须有一个父类。通过extends关键字可以申明一个类是继承外一个类而来的。

##定义

继承是使用已存在的类的定义作为基础建立新类的技术，新类的定义可以增加新的数据或新的功能，可以用父类的功能，但不能选择性地继承父类。通过使用继承我们能够非常方便地复用以前的代码，够大大的提高开发的效率。

1、子类拥有父类非private的属性和方法。

2、子类可以拥有自己属性和方法，即子类可以对父类进行扩展。

3、子类可以用自己的方式实现父类的方法。

讲到继承一定少不了这三个东西：**构造器、protected关键字、向上转型。**

构造器。对于构造器而言，它只能够被调用，而不能被继承。 调用父类的构造方法我们使用super()可。

对于继承而已，子类会默认调用父类的构造器，但是如果没有默认的父类构造器，子类必须要显示的定父类的构造器，而且必须是在子类构造器中做的第一件事(第一行代码)。

protected关键字

private访问修饰符，对于封装而言，是最好的选择，但这个只是基于理想的世界，有时候我们需要这的需求：我们需要将某些事物尽可能地对这个世界隐藏，但是仍然允许子类的成员来访问它们。这个候就需要使用到protected。

对于protected而言，它指明就类用户而言，他是private，但是对于任何继承与此类的子类而言或者他任何位于同一个包的类而言，他却是可以访问的。

诚然尽管可以使用protected访问修饰符来限制父类属性和方法的访问权限，但是最好的方式还是将性保持为private(我们应当一致保留更改底层实现)，通过protected方法来控制类的继承者的访问权。

向上转型

将子类转换成父类，在继承关系上面是向上移动的，所以一般称之为向上转型。由于向上转型是从一叫专用类型向较通用类型转换，所以它总是安全的，唯一发生变化的可能就是属性和方法的丢失。这是为什么编译器在“未曾明确表示转型”活“未曾指定特殊标记”的情况下，仍然允许向上转型的原。

首先我们需要明确，继承存在如下缺陷：

1、父类变，子类就必须变。

2、继承破坏了封装，对于父类而言，它的实现细节对与子类来说都是透明的。

3、继承是一种强耦合关系。

所以说当我们使用继承的时候，我们需要确信使用继承确实是有效可行的办法。那么到底要不要使用承呢？《Think in java》中提供了解决办法：问一问自己是否需要从子类向父类进行向上转型。如果须向上转型，则继承是必要的，但是如果不需要，则应当好好考虑自己是否需要继承。

重写

如果一个类从它的父类继承了一个方法，如果这个方法没有被标记为 final ，就可以对这个方法进行写。

重写的好处是：**能够定义特定于子类类型的行为，这意味着子类能够基于要求来实现父类的方法。**

在面向对象编程中， overriding 意味着去重写父类已经存在的方法。

示例：

```
class Animal{
public void move(){
System.out.println("Animals can move");
}
}
class Dog extends Animal{
public void move(){
System.out.println("Dogs can walk and run");
}
}
public class Test{
public static void main(String args[]){
Animal a = new Animal();
Animal b = new Dog();
a.move();
b.move();
}
}
```

结果：

```
Animals can move
Dogs can walk and run
```

在上面的例子中，你可以看到尽管 b 是 Animal 类型，但它运行了 dog 类的方法。

原因是：在编译时会检查引用类型。然而，在运行时，JVM 会判定对象类型到底属于哪一个对象。因，在上面的例子中，虽然 Animal 有 move 方法，程序会正常编译。在运行时，会运行特定对象的方。

示例：

```
class Animal{
public void move(){
System.out.println("Animals can move");
}
}
class Dog extends Animal{
public void move(){
System.out.println("Dogs can walk and run");
```

```
}
public void bark(){
System.out.println("Dogs can bark");
}
}
public class Test{
public static void main(String args[]){
Animal a = new Animal();
Animal b = new Dog();
a.move();
b.move();
b.bark();
}
}
```

这将产生如下结果：

```
Test.java:30: cannot find symbol symbol : method bark() location: class Animal b.bark();^
```

这个程序在编译时将抛出一个错误，因为 b 的引用类型 Animal 没有一个名字叫 bark 的方法。

方法重写规则

1. 重写方法的参数列表应该与原方法完全相同。
2. 返回值类型应该和原方法的返回值类型一样或者是它在父类定义时的子类型。
3. 重写函数访问级别限制不能比原函数高。举个例子：如果父类方法声明为公有的，那么子类中的重方法不能是私有的或是保护的。
4. 只有被子类继承时，方法才能被重写。
5. 方法定义为 final，将导致不能被重写。
6. 一个方法被定义为 static，将使其不能被重写，但是可以重新声明。
7. 一个方法不能被继承，那么也不能被重写。
8. 和父类在一个包中的子类能够重写任何没有被声明为 private 和 final 的父类方法。
9. 和父类不在同一个包中的子类只能重写 non-final 方法或被声明为 public 或 protected 的方法
10. 一个重写方法能够抛出任何运行时异常，不管被重写方法是否抛出异常。然而重写方法不应该抛比被重写方法声明的更新更广泛的已检查异常。重写方法能够抛出比被重写方法更窄或更少的异常。
11. 构造函数不能重写。

方法覆盖 (Overriding) 和方法重载 (Overloading)

Java 中的方法重载发生在同一个类里面两个或者是多个方法的方法名相同但是参数不同的情况。与此对，方法覆盖是说子类重新定义了父类的方法。方法覆盖必须有相同的方法名，参数列表和返回类型覆盖者可能不会限制它所覆盖的方法的访问

super 关键字

super是用在子类中，目的是访问直接父类中被屏蔽的变量或方法。

子类构造方法中要调用父类的构造方法:

```
super(参数列表)
```

子类引用父类成员变量:

```
super.成员变量名
```

子类成员方法覆盖了父类成员方法时，也就是子类和父类有完全相同的方法定义（但方法体可以不同:

```
super.方法名(参数列表)
```

示例:

```
class Animal{
public void move(){
System.out.println("Animals can move");
}
}
class Dog extends Animal{
public void move(){
super.move();
System.out.println("Dogs can walk and run");
}
}
public class Test{
public static void main(String args[]){
Animal b = new Dog();
b.move();
}
}
```

结果:

```
Animals can move
Dogs can walk and run
```

this关键字

在类的构造方法中，通过this调用另一个构造方法:

```
this(参数列表)
```

方法参数或者方法中的局部变量和成员变量同名的情况下，成员变量被屏蔽，此时要访问成员变量则要用

```
this.成员变量名
```

的方式来引用成员变量。当然，在没有同名的情况下，可以直接用成员变量的名字，可不用this。

在方法中，需要引用当前对象时候。

```
this
```

其实这些用法总结都是从对“this是指向对象本身的一个指针”这句话的更深入的理解而来。