首先请静下心来仔细阅读，这边博客有点绕。好了废话不多说了直接上代码！

static class A {

public String show(D obj) {

return ("A and D");

}

public String show(A obj) {

return ("A and A");

}

}

static class B extends A {

public String show(B obj) {

return ("B and B");

}

public String show(A obj) {

return ("B and A");

}

}

static class C extends B {

}

static class D extends B {

}

static class E extends C {

}

static class F extends E {

}

请问以下输出结果是什么？

A a1 = new A();

A a2 = new B();

B b = new B();

C c = new C();

D d = new D();

F f = new F();

System.out.println("1: " + a1.show(b));

System.out.println("2: " + a1.show(c));

System.out.println("3: " + a1.show(d));

System.out.println("4: " + a2.show(b));

System.out.println("5: " + a2.show(c));

System.out.println("6: " + a2.show(d));

System.out.println("7: " + b.show(b));

System.out.println("8: " + b.show(c));

System.out.println("9: " + b.show(d));

System.out.println("10: " + a1.show(f));

System.out.println("11: " + a2.show(f));2

答案

1: A and A

2: A and A

3: A and D

4: B and A

5: B and A

6: A and D

7: B and B

8: B and B

9: A and D

10: A and A

11: B and A

分析

方法的重写Overriding和重载Overloading是Java多态性的不同表现。重写Overriding是父类与子类之间多态性的一种表现，重载Overloading是一个类中多态性的一种表现。如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写(Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被“屏蔽”了。如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。Overloaded的方法是可以改变返回值的类型。总结得以下规律：

当子类实例指向父类引用（A a2 = new B()）时，是子类决定了父类引用调用谁的成员方法。

1. 如果子类重写了父类的方法，父类引用会优先调用子类重写的方法

2. 如果子类没有重写父类的方法，父类引用会调用自己的方法

当子类实例指向子类引用(B b = new B())时，子类引用优先调用子类的方法，如果子类没有此方法，则会调用父类的方法

　　一句话概括： 当超类对象引用变量引用子类对象时，被引用对象的类型而不是引用变量的类型决定了调用谁的成员方法，但是这个被调用的方法必须是在超类中定义过的，也就是说被子类覆盖的方法。 （但是如果强制把超类转换成子类的话，就可以调用子类中新添加而超类没有的方法了。）

　　好了，先温习到这里，言归正传！实际上这里涉及方法调用的优先问题 ，优先级由高到低依次为：this.show(O)、super.show(O)、this.show((super)O)、super.show((super)O)。让我们来看看它是怎么工作的。

　　比如④，a2.show(b)，a2是一个引用变量，类型为A，则this为a2，b是B的一个实例，于是它到类A里面找show(B obj)方法，没有找到，于是到A的super(超类)找，而A没有超类，因此转到第三优先级this.show((super)O)，this仍然是a2，这里O为B，(super)O即(super)B即A，因此它到类A里面找show(A obj)的方法，类A有这个方法，但是由于a2引用的是类B的一个对象，B覆盖了A的show(A obj)方法，因此最终锁定到类B的show(A obj)，输出为"B and A”。

　　再比如⑧，b.show©，b是一个引用变量，类型为B，则this为b，c是C的一个实例，于是它到类B找show(C obj)方法，没有找到，转而到B的超类A里面找，A里面也没有，因此也转到第三优先级this.show((super)O)，this为b，O为C，(super)O即(super)C即B，因此它到B里面找show(B obj)方法，找到了，由于b引用的是类B的一个对象，因此直接锁定到类B的show(B obj)，输出为"B and B”。

　　按照上面的方法，可以正确得到其他的结果。

　　问题还要继续，现在我们再来看上面的分析过程是怎么体现出蓝色字体那句话的内涵的。它说：当超类对象引用变量引用子类对象时，被引用对象的类型而不是引用变量的类型决定了调用谁的成员方法，但是这个被调用的方法必须是在超类中定义过的，也就是说被子类覆盖的方法。还是拿a2.show(b)来说吧。

　　 a2是一个引用变量，类型为A，它引用的是B的一个对象，因此这句话的意思是由B来决定调用的是哪个方法。因此应该调用B的show(B obj)从而输出"B and B”才对。但是为什么跟前面的分析得到的结果不相符呢？！问题在于我们不要忽略了蓝色字体的后半部分，那里特别指明：这个被调用的方法必须是在超类中定义过的，也就是被子类覆盖的方法。B里面的show(B obj)在超类A中有定义吗？没有！那就更谈不上被覆盖了。实际上这句话隐藏了一条信息：它仍然是按照方法调用的优先级来确定的。它在类A中找到了show(A obj)，如果子类B没有覆盖show(A obj)方法，那么它就调用A的show(A obj)（由于B继承A，虽然没有覆盖这个方法，但从超类A那里继承了这个方法，从某种意义上说，还是由B确定调用的方法，只是方法是在A中实现而已）；现在子类B覆盖了show(A obj)，因此它最终锁定到B的show(A obj)。这就是那句话的意义所在。

————————————————

版权声明：本文为CSDN博主「遗失的少年宫」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/weixin\_36637758/article/details/95089175