什么是多态?

多态分两种：

(1)编译时多态（设计时多态）：方法重载。

(2)运行时多态：JAVA 运行时系统根据调用该方法的实例的类型来决定选择调用哪个方法则被称为运行时多态。（我们平时说得多的事运行时多态，所以多态主要也是指运行时多态）

运行时多态存在的三个必要条件：

一、要有继承（包括接口的实现）；

二、要有重写；

三、父类引用指向子类对象。

详细解释：

运行时多态的解释：

a.运行时多态是指程序中定义的引用变量所指向的具体类型和

b.通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量倒底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定.

1.程序序中定义的引用变量所指向的具体类型不确定(即一个引用变量倒底会指向哪个类的实例对象) 。

例子 ：

driver 类中 drive 方法 （Vehicle 类 vehicle）{}

•oneDriver.drive(new car())

•oneDriver.drive(new bus())

其中 vehicle 变量无法确定具体使用哪个子类实例。

1.通过该引用变量发出的方法调用在编程时并不确定(该引用变量发出的方法调用到底是哪个类中实现的方法) 。

例子 : 厨师，园丁，理发师的 Cut 方法调用.persion.cut().

多态的好处：

1.可替换性（substitutability）。多态对已存在代码具有可替换性。例如，多态对圆Circle类工作，对其他任何圆形几何体，如圆环，也同样工作。

2.可扩充性（extensibility）。多态对代码具有可扩充性。增加新的子类不影响已存在类的多态性、继承性，以及其他特性的运行和操作。实际上新加子类更容易获得多态功能。例如，在实现了圆锥、半圆锥以及半球体的多态基础上，很容易增添球体类的多态性。

3.接口性（interface-ability）。多态是超类通过方法签名，向子类提供了一个共同接口，由子类来完善或者覆盖它而实现的。如图 8.3 所示。图中超类 Shape 规定了两个实现多态的接口方法，computeArea()以及 computeVolume()。子类，如 Circle 和 Sphere 为了实现多态，完善或者覆盖这两个接口方法。

4.灵活性（flexibility）。它在应用中体现了灵活多样的操作，提高了使用效率。

5.简化性（simplicity）。多态简化对应用软件的代码编写和修改过程，尤其在处理大量对象的运算和操作时，这个特点尤为突出和重要。

实际运用：

结合配置文件的使用，联系 Spring 框架，利用反射，动态的调用类，同时不用修改源代码，直接添加新类和修改配置文件，不需要重启服务器便可以扩展程序。

小结：

使用父类类型的引用指向子类的对象，该引用调用的师父类中定义的方法和变量，变量不能被重写（覆盖）；如果子类中重写了父类中的一个方法，那么在调用这个方法的时候，将会调用子类中的这个方法；

注意特殊情况，如果该父类引用所调用的方法参数列表未定义，就调用该父类的父类中查找，如果还没找到就强制向上类型转换参数列表中的参数类型，具体优先级高到低依次如下：

this.show(0)、super.show(0)、this.show((super)0)、super.show((super)0)。

经典笔试题（混合重载和重写）：

（一）相关类

复制代码 代码如下:

```
class A {
                public String show(D obj)...{
                            return ("A and D");
                }
                public String show(A obj)...{
```

```
                        return ("A and A");
        }
}
class B extends A{
        public String show(B obj)...{
                        return ("B and B");
        }
        public String show(A obj)...{
                        return ("B and A");
        }
}
class C extends B...{}
class D extends B...{}
```

（二）问题：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();

System.out.println(a1.show(b));     ①

System.out.println(a1.show(c));     ②

System.out.println(a1.show(d));     ③

System.out.println(a2.show(b));     ④

System.out.println(a2.show(c));     ⑤

System.out.println(a2.show(d));     ⑥

System.out.println(b.show(b));          ⑦

System.out.println(b.show(c));          ⑧

System.out.println(b.show(d));          ⑨
```

（三）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

（四）分析

①②③比较好理解，一般不会出错。④⑤就有点糊涂了，为什么输出的不是"B and B”呢？！！先来回顾一下多态性。

运行时多态性是面向对象程序设计代码重用的一个最强大机制，动态性的概念也可以被说成“一个接口，多个方法”。Java 实现运行时多态性的基础是动态方法调度，它是一种在运行时而不是在编译期调用重载方法的机制。

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但同时参数列表也得不同。

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(0)、super.show(0)、this.show((super)0)、super.show((super)0)。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)0)，this 仍然是 a2，这里 0 为 B，(super)0 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(c)，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)0)，this 为 b，0 为 C，(super)0 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。