# 多态

## 概念

事务存在多种形态，指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同而采用多种不同的行为方式。（发送消息就是函数调用）

父类的引用指向哪个类的实例就调用哪个类中的方法；同样是使用父类的引用，调用同一个名称的方法，却可以得到不同的调用结果，这就是Java中的多态，即：同一函数，多种形态；

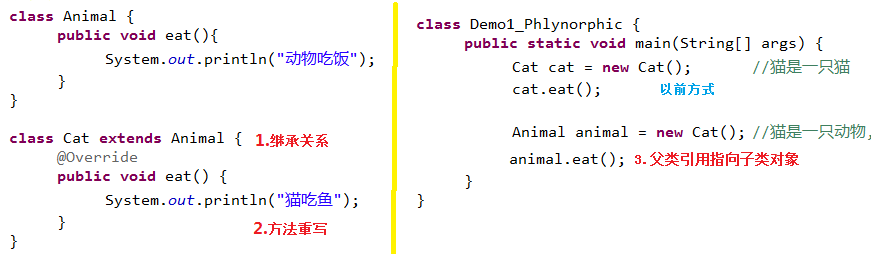
实现方式： Java存在两种多态，重载和重写

## 三个条件

要有继承关系

要有方法重写

要有父类引用指向子类对象



## 好处与弊端

好处：

提高了代码的维护性—继承保证

提高了代码的可扩展性，灵活地扩展我们开发的程序。—多态保证

作用： 消除类型之间的耦合关系

弊端：

提高扩展性，但是只能使用父类引用指向父类成员。不能使用子类新增的属性和行为

## 分类

实际上多态包括动态多态和静态多态。

### 静态多态

静态多态也称为编译时多态，即在编译时决定调用哪个方法；

静态多态一般是指方法重载；

只要构成了方法重载，就可以认为形成了静态多态的条件；

静态多态与是否发生继承没有必然联系。

### 动态多态

动态多态也称为运行时多态，即在运行时才能确定调用哪个方法；

形成动态多态必须具体以下条件：

必须要有继承的情况存在；

在继承中必须要有方法覆盖；

必须由基类的引用指向派生类的实例，并且通过基类的引用调用被覆盖的方法；

由上述条件可以看出，继承是实现动态多态的首要前提。

# 运行时多态

## 多态中的关系--成员访问

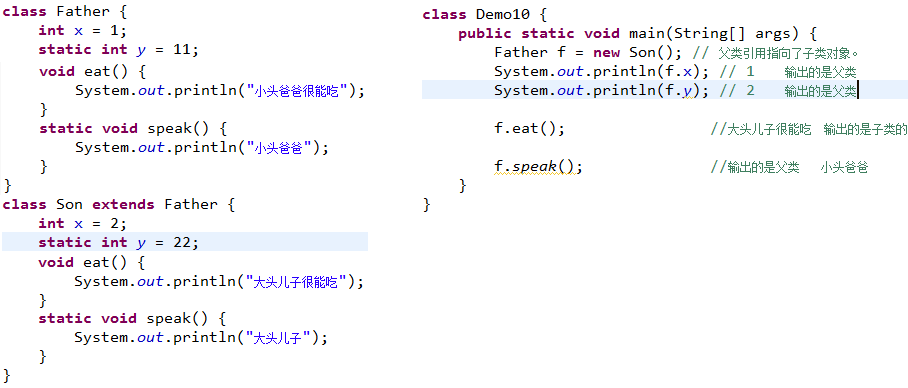
### 编译与运行

编译：找其中的方法，使其程序能通过；

运行：根据编译，看结果

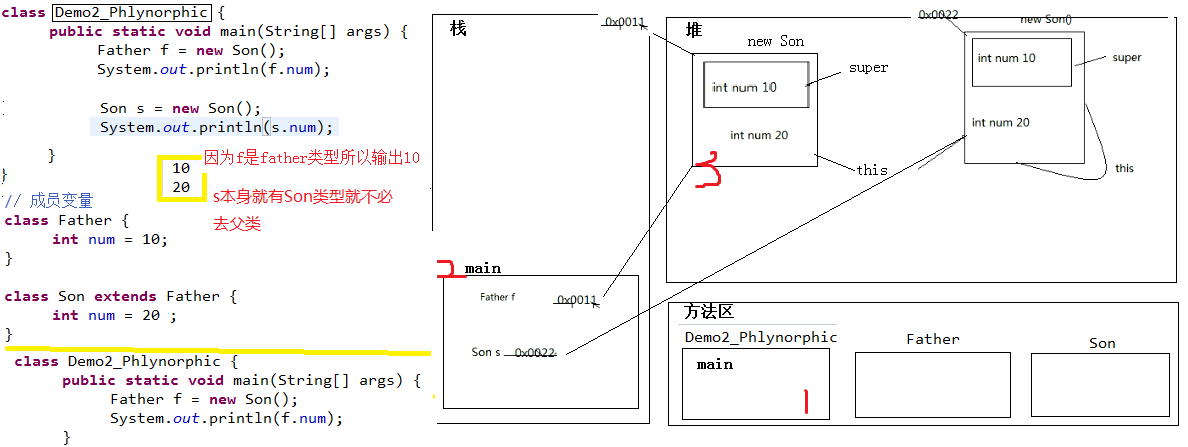


所以前面我说静态方法不能算方法的重写



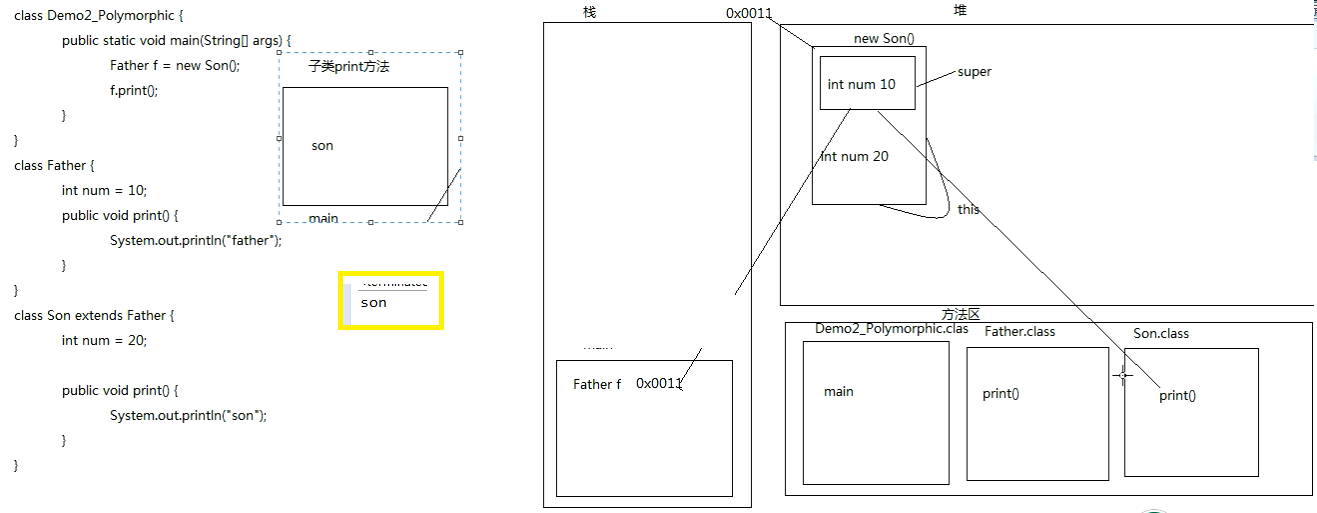
### 成员变量

编译看左边（父类），运行看左边（父类）



### 实例方法

编译看左边（父类），运行看右边（子类）

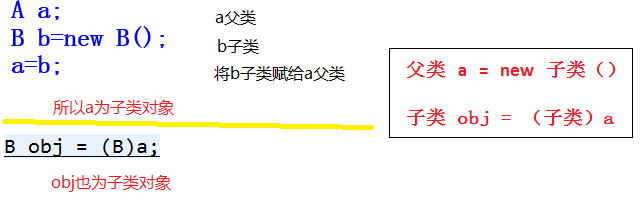


### 静态方法

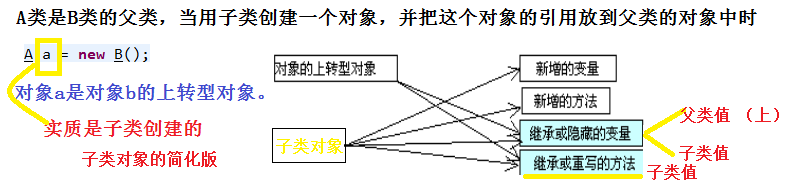
编译看左边（父类），运行看左边（父类）

相当于类名调用

## 对象转型



### 上转型对象



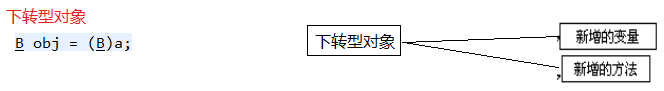
所以子类的值可以直接赋值给父类

ActivityPo activityPo = activityQueryVo;*//上转型对象：所以子类的值可以直接赋值给父类*

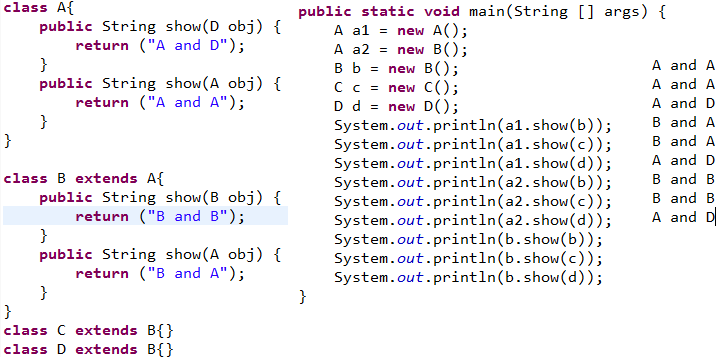
### 下转型对象

强制类型转化。

可以使用instanceof来判断某引用型变量所“指向”的对象是否属于该类或该类的子类。例:a instanceof B



## [子类对象父类可用的原则](http://0.0.0.1/)



1. 子类拥有父类的所有方法

比如B b = new B(),就会得到3（A1 + B2）个方法

上转型对象：父类的 + 子类重写的 （除去子类新增）

1. [子类对象父类可用的原则](http://0.0.0.1/)()

参数问题：参数是b，当b匹配不到时，缩小为父类