IDEA 的快捷键

1)删除当前行,

默认是 ctrl + Y 自己配置 ctrl + d

2)

复制当前行,

自己配置 ctrl + alt + 向下光标

3)

补全代码 alt + /

4)

添加注释和取消注释 ctrl + / 【第一次是添加注释，第二次是取消注释】

5)

导入该行需要的类 先配置 auto import，然后使用 alt+enter 即可

6)

快速格式化代码 ctrl + alt + L

7)

快速运行程序

自己定义 alt + R

8)

生成构造器等

alt + insert [提高开发效率]

9)

查看一个类的层级关系 ctrl + H [学习继承后，非常有用]

10)

将光标放在一个方法上，输入 ctrl + B，可以定位到方法 [学继承后，非常有用]

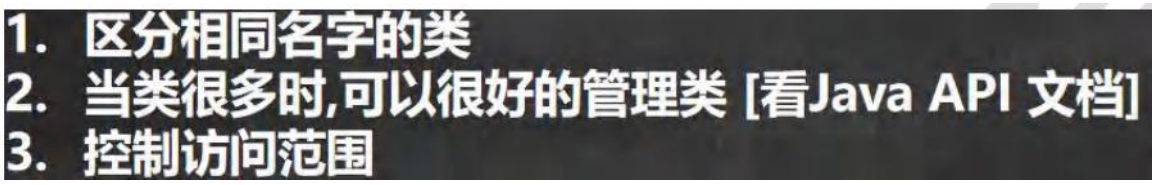
11)

自动的分配变量名 ，通过 在后面加 .var [老师最喜欢的]

12)

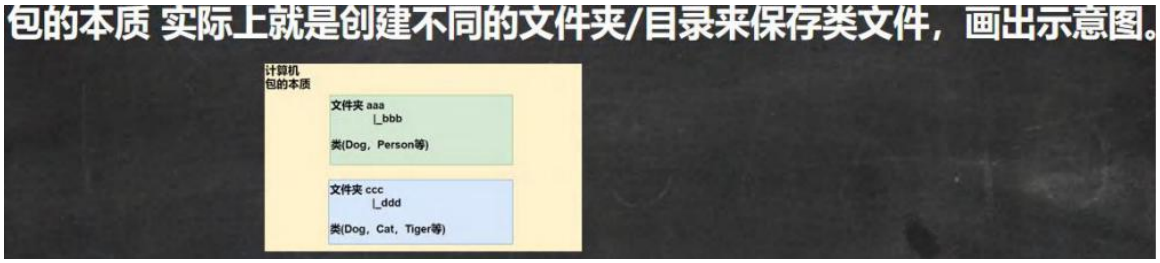
还有很多其它的快捷键

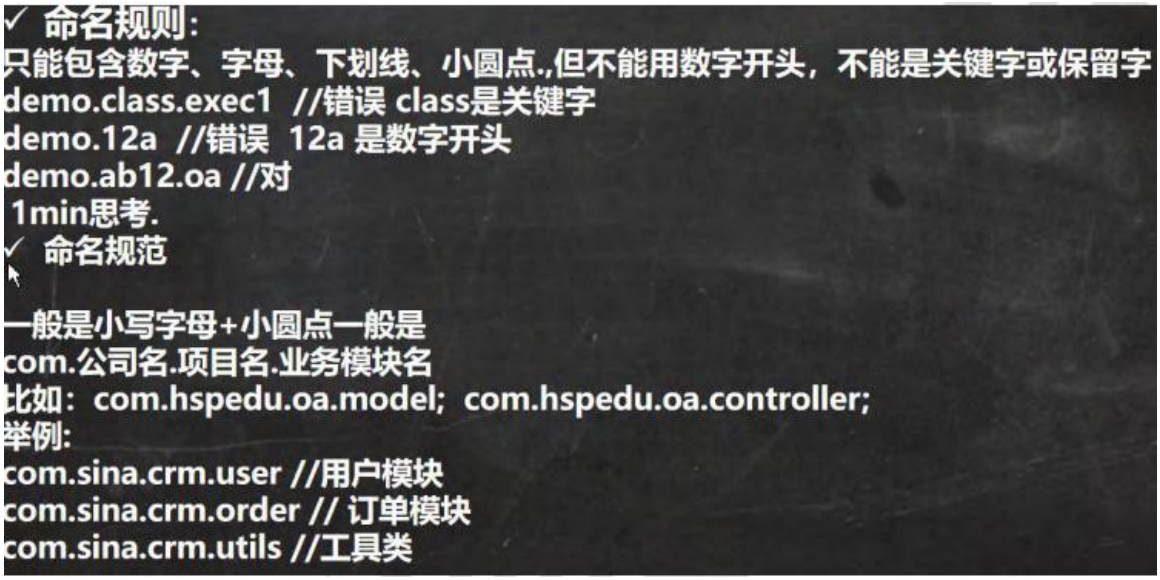
包的作用



基本语法

包的本质





一个包下,包含很多的类,java 中常用的包有:

1)

java.lang.* //lang 包是基本包，默认引入，不需要再引入.

2)

java.util.* //util 包，系统提供的工具包, 工具类，使用 Scanner

3)

java.net.* //网络包，网络开发

4)

java.awt.* //是做 java 的界面开发，GUI

修饰符

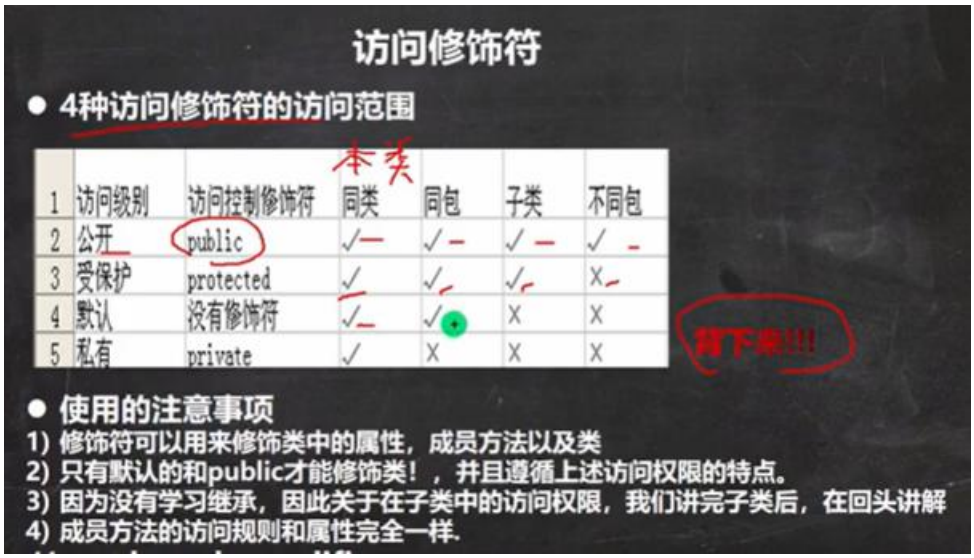
java 提供四种访问控制修饰符号，用于控制方法和属性(成员变量)的访问权限（范围）:

1) 公开级别:用 public 修饰,对外公开

2) 受保护级别:用 protected 修饰,对子类和同一个包中的类公开

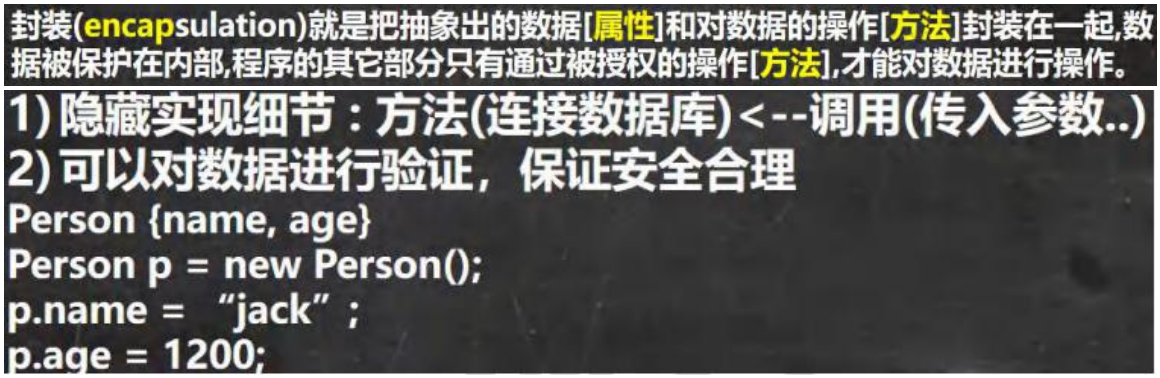
3) 默认级别:没有修饰符号,向同一个包的类公开.

4) 私有级别:用 private 修饰,只有类本身可以访问,不对外公开.

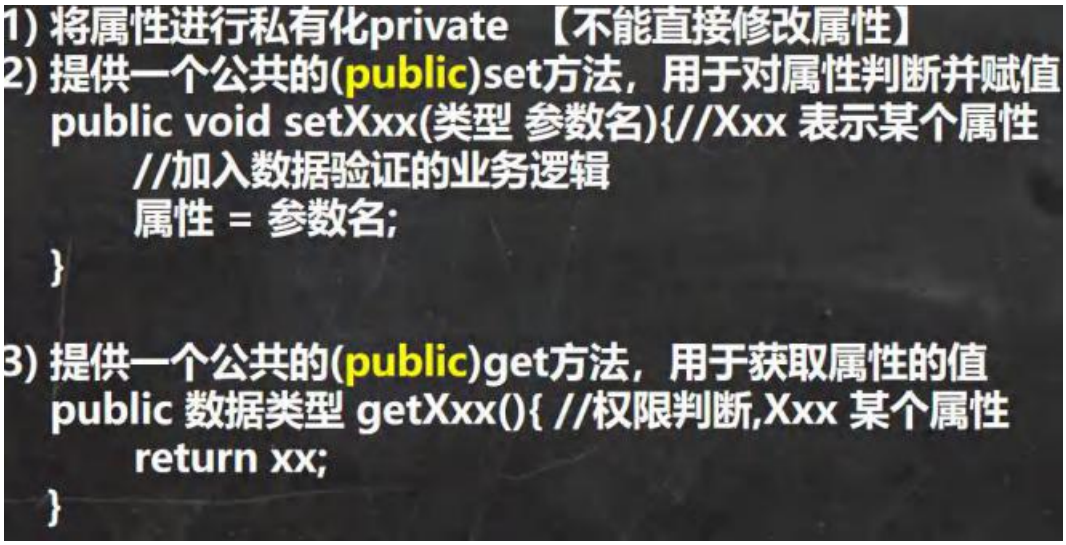


面向对象编程三大基本特征：

封装、继承、多态



步骤

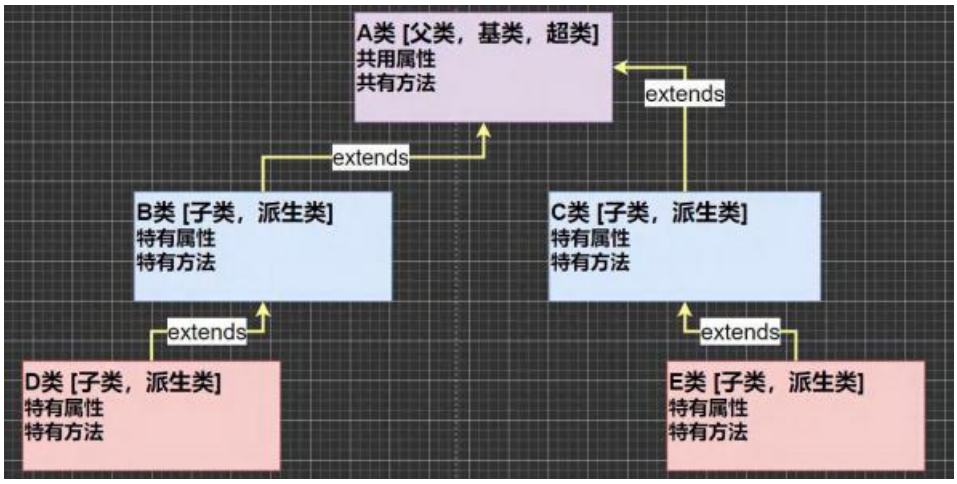


Set get 设置的防护机制 如果直接使用构造器 防护机制就没用了 但是可以将 set 加入到构造器中

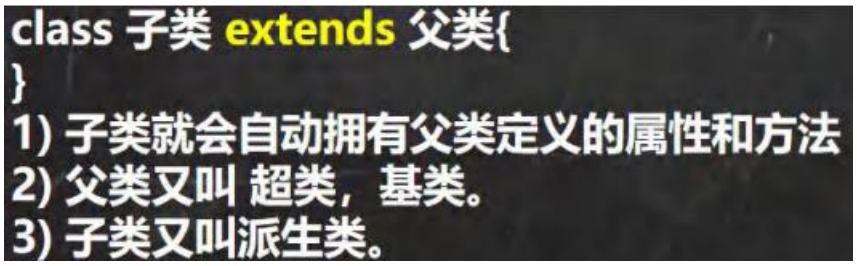
继承

为啥要继承？　代码复用性

继承可以解决代码复用,让我们的编程更加靠近人类思维.当多个类存在相同的属性(变量)和方法时,可以从这些类中 抽象出父类,在父类中定义这些相同的属性和方法，所有的子类不需要重新定义这些属性和方法，只需要通过 extends 来 声明继承父类即可。



语法



细节：

1) 子类继承了所有的属性和方法，非私有的属性和方法可以在子类直接访问, 但是私有属性和方法不能在子类直接访问，要通过父类提供公共的方法去访问

2) 子类必须调用父类的构造器， 完成父类的初始化

3) 当创建子类对象时，不管使用子类的哪个构造器，默认情况下总会去调用父类的无参构造器，如果父类没有提供无 参构造器，则必须在子类的构造器中用 super 去指定使用父类的哪个构造器完成对父类的初始化工作，否则，编译不会通过(怎么理解。)［举例说明］

4) 如果希望指定去调用父类的某个构造器，则显式的调用一下 : super(参数列表)

5) super 在使用时，必须放在构造器第一行(super 只能在构造器中使用)

6) super() 和 this() 都只能放在构造器第一行，因此这两个方法不能共存在一个构造器

7) java 所有类都是 Object 类的子类, Object 是所有类的基类.

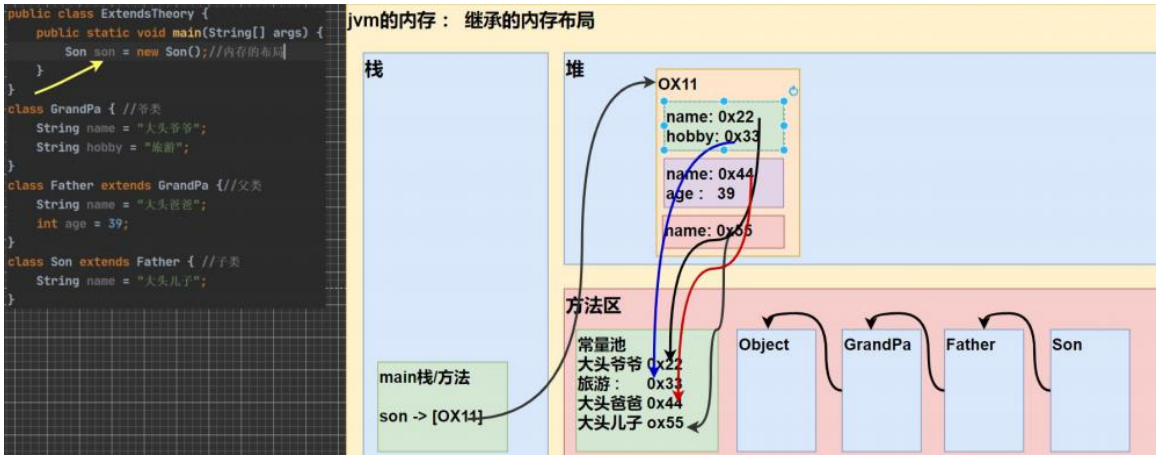
8) 父类构造器的调用不限于直接父类！将一直往上追溯直到 Object 类(顶级父类)

9) 子类最多只能继承一个父类(指直接继承)，即 java 中是单继承机制。

思考：如何让 A 类继承 B 类和 C 类？ 【A 继承 B， B 继承 C】

10) 不能滥用继承，子类和父类之间必须满足 is-a 的逻辑关系

继承本质 在方法区从上往下



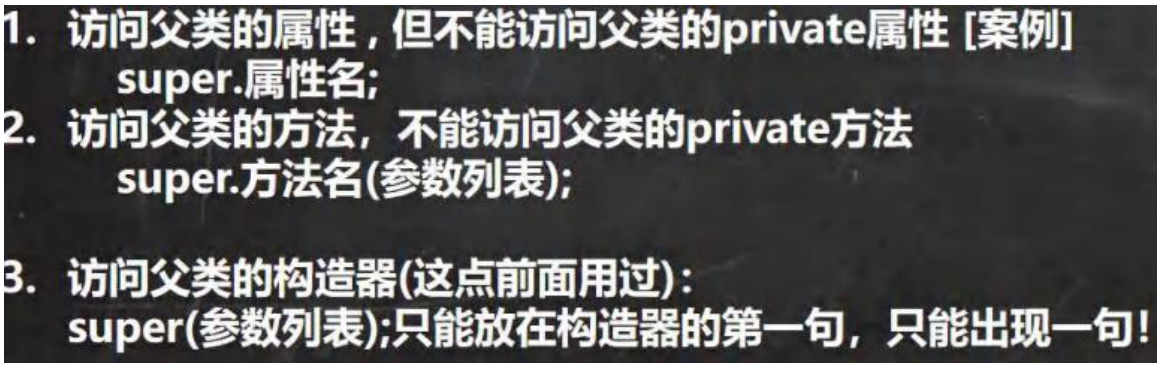
创建子类先加载父类

访问时从下往上找（就近）

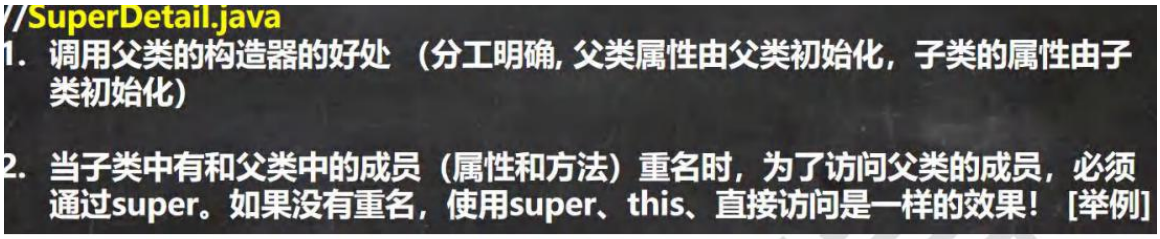
```
Son son = new Son();//内存的布局
//?-> 这时请大家注意，要按照查找关系来返回信息
//(1) 首先看子类是否有该属性
//(2) 如果子类有这个属性，并且可以访问，则返回信息
//(3) 如果子类没有这个属性，就看父类有没有这个属性(如果父类有该属性，并且可以访问，就返回信息..)
//(4) 如果父类没有就按照(3)的规则，继续找上级父类，直到Object...
System.out.println(son.name);
```

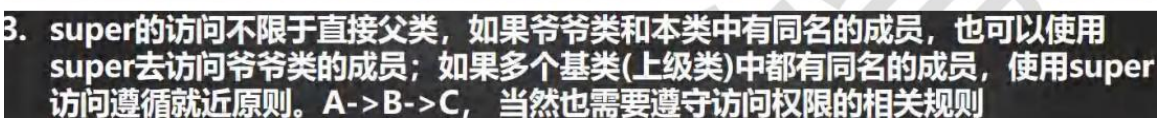
Super

super 代表父类的引用，用于**访问父类的属性、方法、构造器**



好处





No.	区别点	this	super
1	访问属性	访问本类中的属性，如果本类没有此属性则从父类中继续查找	从父类开始查找属性
2	调用方法	访问本类中的方法,如果本类没有此方法则从父类继续查找.	从父类开始查找方法
3	调用构造器	调用本类构造器，必须放在构造器的首行	调用父类构造器，必须放在子类构造器的首行
4	特殊	表示当前对象	子类中访问父类对象

子类调用方法时的顺序

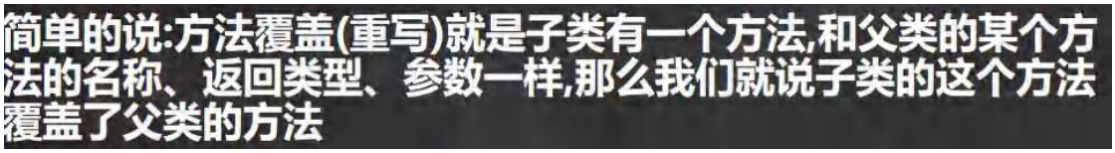
```
public void sum() {
    System.out.println("B类的sum()");
    //希望调用父类-A 的cal方法
    //这时，因为子类B没有cal方法，因此我可以使用下面三种方式

    //找cal方法时，顺序是：
    // (1)先找本类，如果有，则调用
    // (2)如果没有，则找父类(如果有，并可以调用，则调用)
    // (3)如果父类没有，则继续找父类的父类,整个规则，就是一样的,直到 Object类
    // 提示：如果查找方法的过程中，找到了，但是不能访问， 则报错
    //       如果查找方法的过程中，没有找到，则提示方法不存在
    cal();
```

其中 this.cal()与 cal(),完全一样地查找

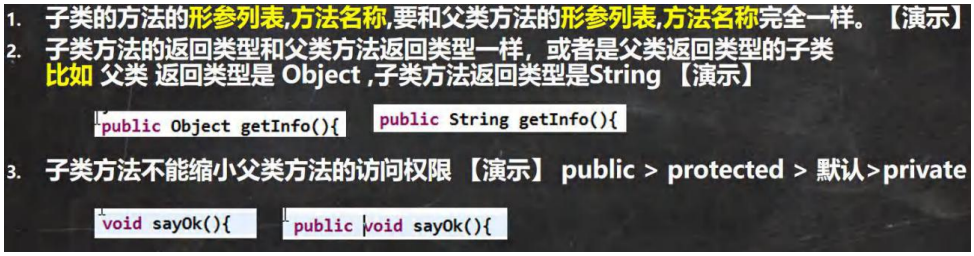
Super 直接跳过（1）直接找父类

属性的访问同方法

方法覆盖 重写 override



也可以覆盖爷爷类



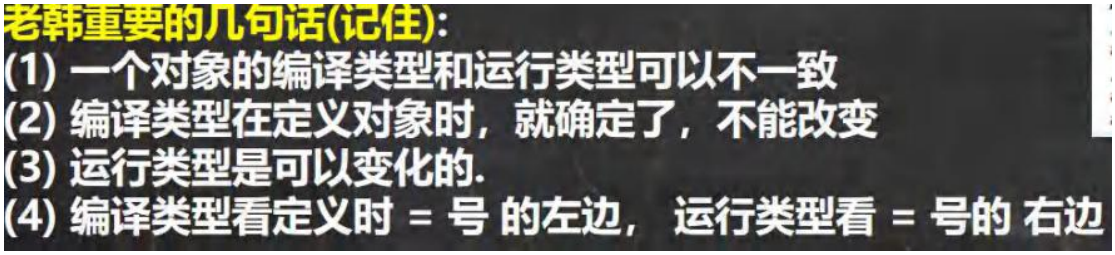
其中方法返回类型 Object 和 String 前者在父类后者在子类 不能颠倒 即父类那个包含子类

访问权限子类可大于父类但不能小于

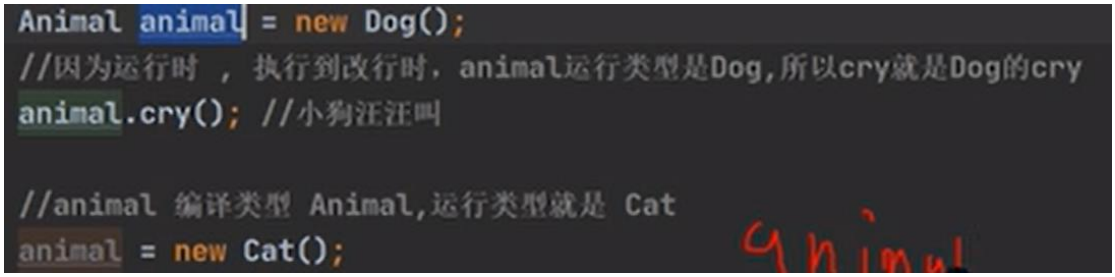
名称	发生范围	方法名	形参列表	返回类型	修饰符
重载(overload)	本类	必须一样	类型，个数或者顺序至少有一个不同	无要求	无要求
重写(override)	父子类	必须一样	相同	子类重写的方法，返回的类型和父类返回的类型一致，或者是其子类	子类方法不能缩小父类方法的访问范围.

多态

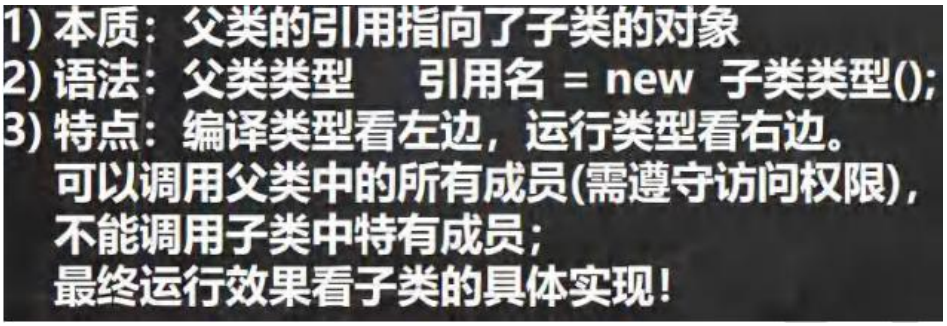
重载重写就体现多态

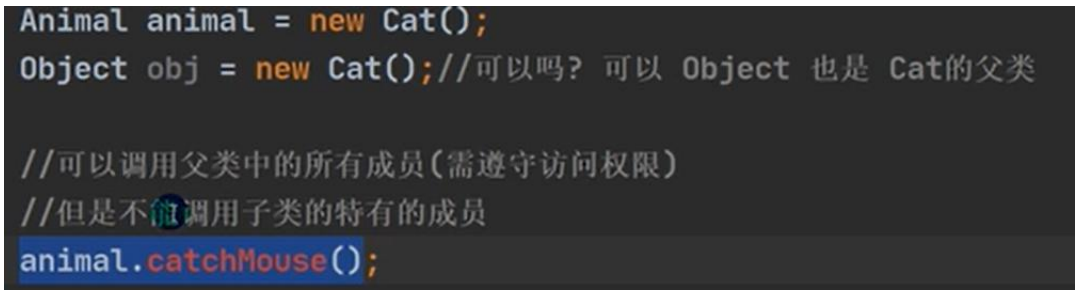


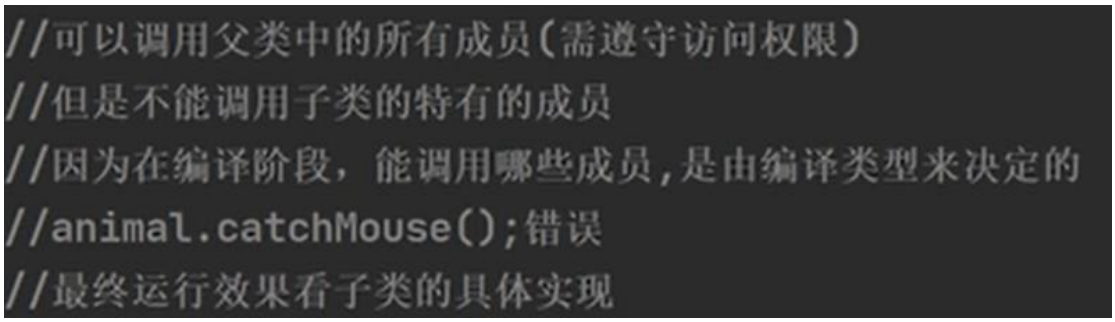
例如



多态向上转型

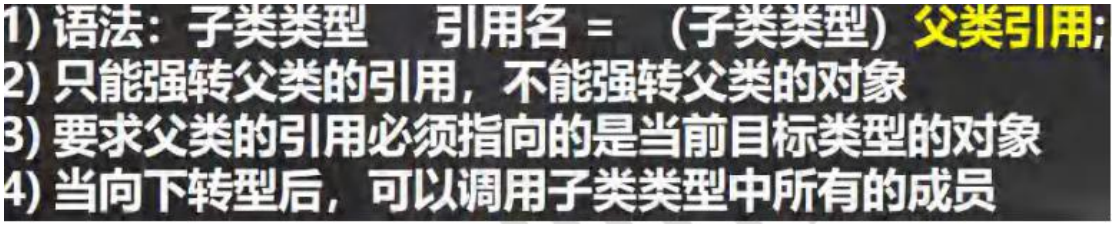






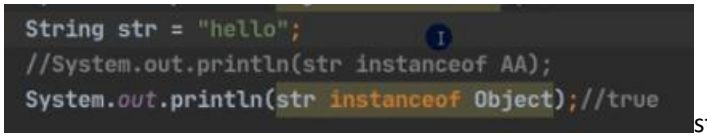
向上转型调用的方法是看运行类型的，从运行类型来向上查找。

向下转型

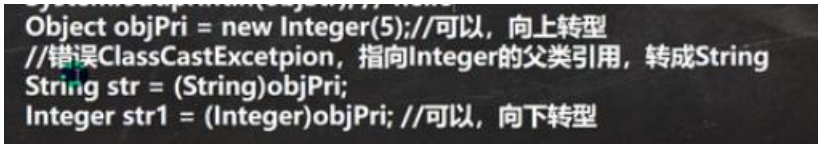


对于属性来说 只看编译类是谁

InstanceOf

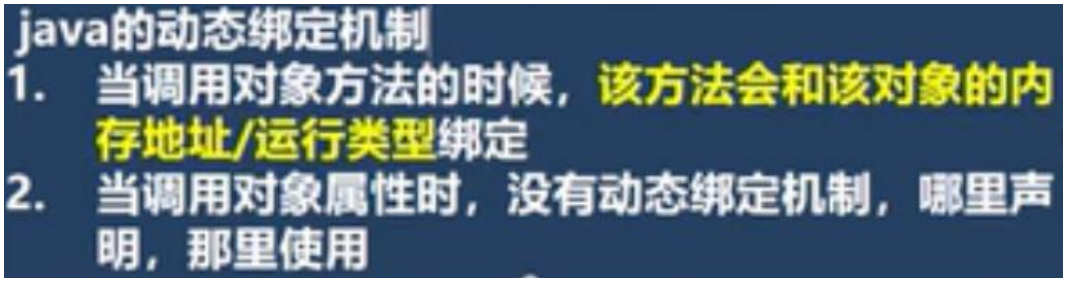


str 字符串的运行类型属于 Object



向下转型时要注意 先前向上转型的运行类型 如上面的 objPri

动态绑定

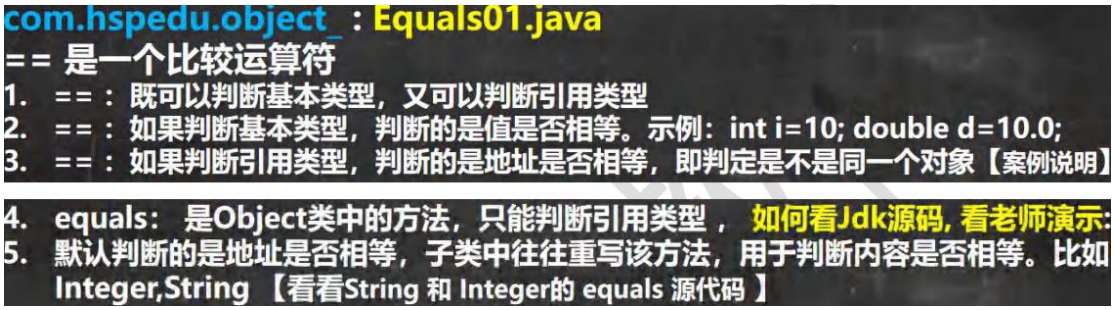


多态数组

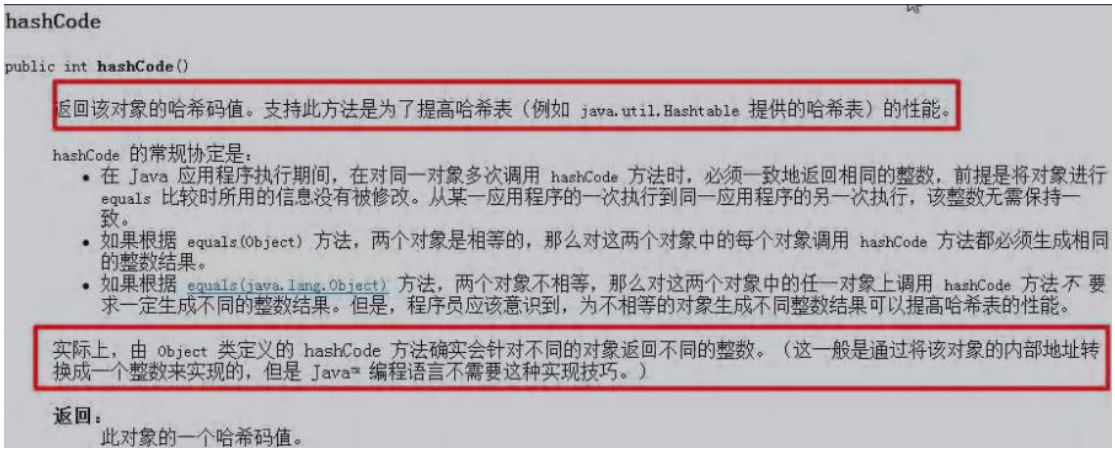
定义类型为父类类型，里面保存的实际元素类型为子类类型

多态参数

Object 详解



Hascode



1) 提高具有哈希结构的容器的效率！
2) 两个引用，如果指向的是同一个对象，则哈希值肯定是一样的！
3) 两个引用，如果指向的是不同对象，则哈希值是不一样的
4) 哈希值主要根据地址号来的！， 不能完全将哈希值等价于地址。
5) 案例演示[HashCode_.java]: obj.hashCode() [测试： A obj1 = new A(); A obj2 = new A(); A obj3 = obj1]
6) 后面在集合，中 hashCode 如果需要的话，也会重写, 在讲解集合时，老韩在说如何重写 hashCode()

toString 方法

1) 基本介绍
默认返回：全类名+@+哈希值的十六进制，【查看 Object 的 toString 方法】
子类往往重写 toString 方法，用于返回对象的属性信息
2) 重写 toString 方法，打印对象或拼接对象时，都会自动调用该对象的 toString 形式.
3) 当直接输出一个对象时，toString 方法会被默认的调用, 比如 System.out.println(monster)； 就会默认调用 monster.toString()

finalize 方法

1) 当对象被回收时，系统自动调用该对象的 finalize 方法。子类可以重写该方法，做一些释放资源的操作【演示】
2)
什么时候被回收：当某个对象没有任何引用时，则 jvm 就认为这个对象是一个垃圾对象，就会使用垃圾回收机制来
销毁该对象，在销毁该对象前，会先调用 finalize 方法。
3)
垃圾回收机制的调用，是由系统来决定(即有自己的 GC 算法), 也可以通过 System.gc() 主动触发垃圾回收机制，测
试：Car [name]
一般当对象为空时，可回收对象释放资源