

向上转型和动态绑定机制

知识梳理

1. 向上转型的概念：

向上转型又叫自动转型、隐式转型。向上转型就是父类引用指向子类实例，也就是子类的对象可以赋值给父类的对象。

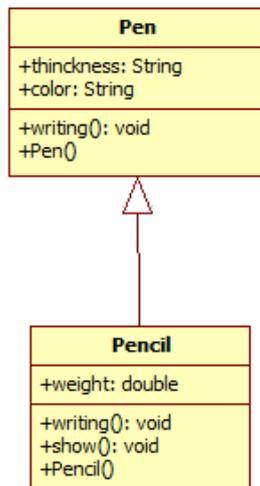
如：`Animal dog=new Dog();` //Dog 类是 Animal 类的子类

注：向上转型是安全的，因为任何子类都继承并接受了父类的方法。从例子中也可以理解，所有的狗狗都属于狗的父类——动物，这是可行的，但是向下转型则不行，若说所有的动物都是狗狗就不成立了。（所以向下转型要通过强制类型转换，后续章节中会讲解）

2、向上转型的应用

1) 当一个子类对象向上转型父类类型以后，就被当成了父类的对象，所能调用的方法会减少，只能调用子类重写了父类的方法以及父类派生的方法（如 `set()`、`get()`方法），而不能调用子类独有的方法。

例如：



对象实例化：

```
Pen pc=new Pencil();//向上转型
```

此时，调用方法调用时

```
pc.writing();  
pc.show();
```

可以调用子类重写父类的方法 writing(),但调用子类独有的方法 show()时就是无效的

2) 父类中的静态方法是不允许被子类重写的。

如父类 Pen 中含有静态方法 draw()

```
public static void draw(){  
    System.out.println("笔可以用来画画");  
}
```

当子类 Pencil 中也定义同名方法时，此时 draw()算 Pencil 类自己独有的方法：

```
public static void draw(){  
    System.out.println("铅笔用来素描");  
}
```

测试类中调用时：

```
Pen pc=new Pencil();//向上转型  
pc.draw();
```

返回结果为：

笔可以用来画画

也就是实际上调用的是父类的静态方法 draw()

知识扩展

多态的实现可以通过向上转型和动态绑定机制来完成,向上转型实现了将子类对象向上转型为父类类型,而动态绑定机制能识别出对象转型前的类型,从而自动调用该类的方法,两者相辅相成。

动态绑定：

绑定就是将一个方法调用同一个方法所在的类连接到一起就是绑定。绑定分为静态绑定和动态绑定两种。

静态绑定：在程序运行之前进行绑定（由编译器和链接程序完成的），也叫做前期绑定。

动态绑定：在程序运行期间由 JVM 根据对象的类型自动的判断应该调用哪个方法，也叫做后期绑定。

静态绑定的例子：

如有一类 Human,它派生出来三个子类 Chinese 类、American 类和 British 类，三个子类中都重写了父类中的方法 `speak() : void`,在测试类中用静态绑定的方式调用方法 `speak()`。

```
Chinese c =new Chinese();
c.speak();
American a =new American();
a.speak();
British b =new British();
b.speak();
```

这种调用方式是在代码里指定的，编译时编译器就知道 c 调用的是 Chinese 的 `speak()`,a 调用的是 American 的 `speak()`。

动态绑定的例子：

如果我们在测试类中做以下改动：

```
//生成父类对象数组,数组长度为5
Human[] human=new Human[5];

int n;
for(int i=0;i<human.length;i++){
    n=(int)(Math.random()*3);//随机产生从0到2中一个数
    switch(n){
        case 0:human[i]=new Chinese();break;
        case 1:human[i]=new American(); break;
        case 2:human[i]=new British(); break;
    }
}
//循环输出,循环体中每个对象分别调用speak()方法
for(int i = 0;i< human .length;i++){
    human[i].speak();
}
```

此时，Human 类中随机生成 Chinese 类、American 类和 British 类的对象，编译器不能根据代码直接确定调用哪个类中的 speak() 方法，直到运行时才能根据产生的随机数 n 的值来确定 human[i] 到底代表哪一个子类的对象，这样才能最终确定调用的是哪个类中的 speak() 方法，这就是动态绑定。

