**Day09-【继承、super、this、抽象类】**

**【继承】**

**一、基本要素**

定义：子类继承父类的属性和行为，使得子类对象具有与父类相同的属性、相同的行为。父类（基类/超类）更通用，子类（派生类）更具体。子类可以直接访问父类中的非私有的属性和行为。

特点:

1. 子类可以拥有父类的“内容”
2. 子类还可以拥有自己专有的内容

好处：

1. 提高代码的复用性。(共性抽取)
2. 类与类之间产生了关系，是多态的前提。

格式:

class 父类{

…

}

Class 子类 extends 父类{

…

}

**二、成员变量**

1.若子类父类中出现不重名的成员变量，此时访问无影响。

2.若子类父类中出现重名的成员变量，可以：

①直接通过子类对象访问成员变量；

（等号左边是谁，则优先用谁，没有则向上找）

Zi zi=new Zi();输出 zi.num，则输出的是子类的num。

②间接通过成员方法访问成员变量。 （该方法属于谁，则优先用谁，没有则往上找）

zi.methodZi();//在methodZi()方法中输出num，则此方法输出的是子类中的num

zi.methodFu();//在methodFu()方法中输出num,则此代码输出的是父类中的num

1. 区分子类方法中重名的三种变量

num;//子类方法中的局部变量

this.num;//子类中的成员变量

super.num;//父类中的成员变量

**三、成员方法**

1.访问特点：在父子类的继承关系中，创建子类对象，创建的对象是谁，就优先用谁，如果没有则向上找。

Zi zi=new Zi();//zi为子类对象

zi.method();//优先用子类中的method，若子类中没有method则用父类中的method。

1. 覆盖重写（override）：在继承关系中，方法的名称一样，参数列表也一样。

与重载（overload）的区别：方法的名称一样，参数列表不一样。

* 1. 特点：创建的是子类对象，则优先用子类方法
  2. 注意事项：

必须保证父子类中方法的名称相同，参数列表也相同。（其中，可以在子类方法前一行写“@override”来检验子类覆盖重写是否达标）

子类方法的【返回值范围】必须【小于等于】父类方法的返回值范围。（String<Object

子类方法的权限必须【大于等于】父类方法的权限修饰符。

扩展提示：public >protected >(default) >private

(default)指的是什么都不写

* 1. 应用场景:对于已经使用的类，尽量不进行修改。推荐定义一个新的类，来重复利用其中共性内容，并且添加改动新内容。（类似手机升级版本）

格式：子类方法覆盖重写时，若想重用父类中方法的内容，可以super.method(); 此行代码后再加入自己新的功能代码。

1. 构造方法

子类构造方法当中默认隐含一个“super()”调用，即一定先调用父类无参构造，再执行子类构造。因此：当父类中只有有参构造时，子类无参构造中必须手动添加super(父类有参构造中的参数类型值),否则程序报错。如super(10);

super的父类构造调用，必须时子类【构造方法】中的第一个语句，且是唯一一个。（即不能一个子类构造调用多次super构造）

【super】关键字的三种用法:

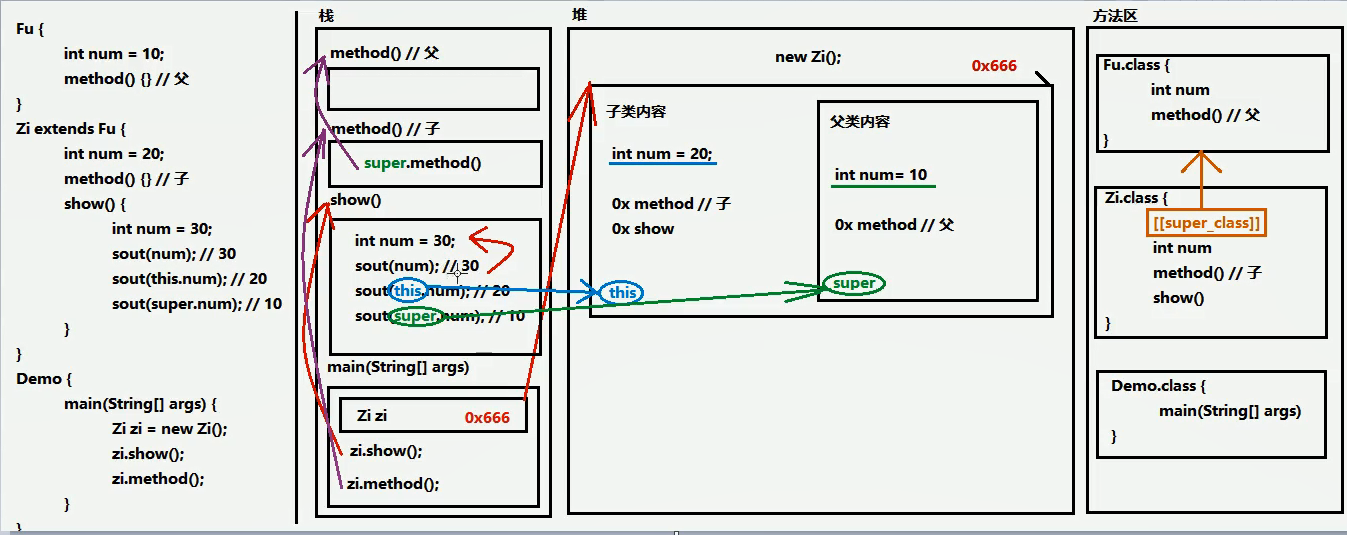
1. 在子类的成员方法中，访问父类的成员变量。（super.num;）
2. 在子类的成员方法中，访问父类的成员方法。(super.method();)

3.在子类的构造方法中，访问父类的构造方法。(super();)

Tips:super关键字是用来访问父类内容的

【this】关键字的三种用法:

1. 在本类的成员方法中，访问本类的成员变量。（在本类方法中直接输出num,即为方法中的局部变量）。若是this.num;即输出本类中的成员变量。
2. 在本类的成员方法中，访问本类的另一个成员方法。（在methodB()方法中写this.methodA();）
3. 在本类的构造方法中，访问本类的另一个构造方法。（如在Zi（）无参构造函数中，写this(123);调用本类中的有参构造。但是this语句只能用一次，如果接着再写this(1,2)）;则会报错
4. super和this两种方法不能同时使用

【super和this关键字内存图解】

【Java语言继承的特点】

1. 单继承。一个类的直接父类只能有唯一一个。

class D extends A,B{} //错误写法，避免产生继承冲突。

1. 多级继承。即一个类可以有父类，“爷爷”，“祖爷爷”等。
2. 一个父类可以拥有很多个子类。

【抽象类】

1.如果父类当中的方法不确定如何进行{}方法体实现，那么这就是一个抽象方法。

2.抽象方法举例：public **abstract** void eat();【抽象方法所在类必须是抽象类】

1. 使用方法：
   1. 不能直接创建new抽象类对象
   2. 必须用一个子类来继承抽象父类
   3. 子类必须覆盖重写抽象父类当中所有的抽象方法。（即子类中去掉抽象方法的abstract，补上方法体大括号。）
   4. 创建子类对象进行使用。
2. 注意事项：
   1. 抽象类不能创建对象，只能创建其非抽象类子类的对象。
   2. 抽象类中可以有构造方法，但是不可以通过new类对象来访问（如Fu是抽象类，fu.Fu();//错误写法！ 可以通过子类的构造方法来访问，因为子类的构造方法默认产生super();详情看第4页super用法。）
   3. 抽象类中不一定包含抽象方法，但有抽象方法的类必定是抽象类。
   4. 抽象类的子类必须覆盖重写父类中所有的抽象方法。否则子类页必须是抽象类。

【综合案例：发红包】

1. 确定需要用到的父类/子类。
2. 确定不同类中用到的方法：返回值类型，方法名称，参数列表。

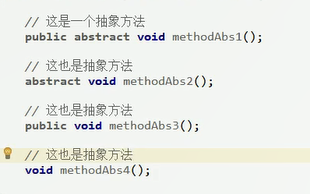
**Day10-【接口、多态】**

接口就是多个类公共的规范，是一种引用数据类型。

1.定义格式：public interface 接口名称{ //接口内容}

备注:虽然关键字是interface，但编译生成的字节码文件仍然是java🡪.class

2.接口中可以包含的内容：常量、抽象方法（Java7及以上）；默认方法、静态方法（Java8及以上）;私有方法（Java9及以上）

3.接口当中的抽象方法，修饰符必须是两个固定的关键字：public abstract （可以选择性忽略不写，但不能写其它的）；方法的三要素可以随意定义。

3.接口使用步骤：

①接口不能直接使用，必须有一个“实现类”来实现该接口(与继承中的子类类似)。

格式：public class 实现类名称 implements 接口名称{

//…}

②接口的实现类必须覆盖重写（实现）接口中所有的抽象方法。（快捷键：再实现类名该行按“Ctrl+enter”+ok一键覆盖重写所有抽象方法）

去掉abstract关键字，加上方法大括号。

注意事项:如果实现类并没有覆盖重写接口中所有的抽象方法，那么这个实现类自己就必须是抽象类。

③创建实现类的对象，进行使用。

4．接口中默认方法的定义

格式:public default 返回值类型 方法名称（参数列表）{ 方法体 }

备注：接口中的默认方法，可以解决接口升级的问题。

升级问题：即在很多实现类覆盖重写了接口中的抽象方法之后，想继续给接口添加抽象方法，但是不希望在每个实现类中都重复进行覆盖重写，便可以把【新添加的抽象方法改成default类型】。

想在实现类中调用默认方法，若实现类中没有，则会向上找接口。

1. 接口的默认方法，可以通过接口实现类对象，直接调用。
2. 接口的默认方法，也可以被接口实现类进行覆盖重写。
3. 接口中静态方法的定义

格式：public static 返回值类型 方法名称（参数列表）{方法体}

注意事项：

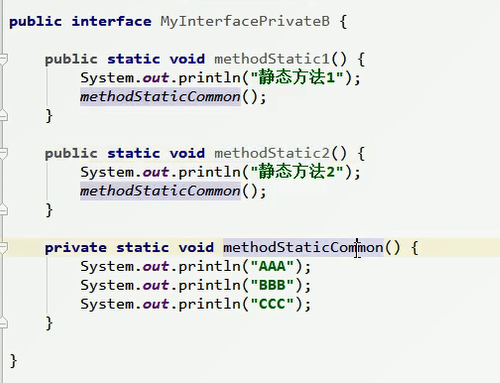
1. 不能通过接口实现类的对象来调用接口当中的静态方法。（如 接口实现类类名称.方法名（）；//错误写法）

正确用法：通过接口名称，直接调用其中的静态方法。格式:接口名称.静态方法名（参数）；

1. 接口的私有方法定义
   1. 普通私有方法（用来解决同一个实现类中多个默认方法之间重复的代码）

格式：private 返回值类型 方法名称（参数列表）{方法体}

* 1. 静态私有方法（解决多个静态方法之间重复代码的问题）

格式：private static 返回值类型 方法名称（参数列表）{方法体}

1. 接口的常量定义和使用

接口中可以定义“成员变量”，但必须使用public static final 三个关键字进行修饰。（其中这三个关键字都可以省略不写）

格式：

public static final 数据类型 常量名称=数据值；

注意：

1. 一旦使用final关键字进行修饰，说明不可改变。
2. 接口中的常量，必须进行赋值，不能不赋值。（int num;//没赋值，错误写法！）
3. 接口中的变量，其名称一般全用大写，若有多个单词则用下划线连接。
4. 使用方法：接口名称.成员变量名称
5. 使用接口需要注意的点

接口没有静态代码块或者构造方法的。

一个类的直接父类是唯一的，但是一个类可以同时实现多个接口。

格式:

public class 实现类名称 implements 接口A，接口B{//覆盖重写所有抽象方法}

①如果实现类所实现的多个接口当中，存在重复的对象方法，那么只需要覆盖重写一次即可。

如果实现类没有覆盖重写所有接口当中的所有抽象方法，那么实现类就必须是一个抽象类。

②如果实现类所实现的多个接口当中，存在重复的默认方法，那么实现类一定要对冲突的默认方法进行覆盖重写。

1. 一个类如果直接父类当中的方法和接口当中的方法产生了冲突，优先使用父类当中的方法。（在Java实现当中，继承优先于接口实现）
2. 接口之间的多继承
   1. 类与类之间是单继承的。直接父类只有一个。
   2. 类与接口之间是多实现的。一个类可以实现多个接口。
   3. 接口与接口之间是多继承的。

注意事项：

1. 多个父接口当中的抽象方法如果重复，没关系。
2. 多个父接口当中的默认方法如果重复，那么子接口必须进行默认方法的覆盖重写，而且带着default关键字。

【多态】

1. 定义：指统一行为，具有多个不同的表现形式。

代码中体现多态性，即父类引用指向子类对象

格式：父类名称 对象名=new 子类名称（）；（若对象名.method ;则调用子类中的method）

或者：接口名称 对象名=new 实现类名称（）；

1. 多态中成员变量的使用特点：

访问成员变量的两种方式：

* 1. 直接通过对象名称访问成员变量：看等号左边是谁，优先用谁，没有则向上找。
  2. 间接通过成员方法访问成员变量：看该方法属于谁，优先用谁，没有则向上找。（即若子类覆盖重写了，则用子；若子类没有覆盖重写，则用父。）

1. 多态中成员方法的使用特点：

在多态的代码中，成员方法的访问规则：看new的是谁，就优先用谁，没有则向上找。（如：

Fu obj =new Zi();

obj.method();//若父子都有method（），则优先用子

obj.methodFu();//子类没有，父类有，向上找父类中的methodFu（）。）

口诀：编译看左边，运行看右边。

（如obj.methodZi();//编译报错 原因：看等号左边是Fu,Fu中没有methodZi(),所以编译报错； 再如obj.method();//运行通过，先看左，Fu中有method所以程序中没有出现红色（没有编译报错），运行看右，子类中也有method（）（若子类没有，则向上找父类），故运行通过）

对比以下:成员变量，编译看左边，运行还看左边；成员方法，编译看左边，运行看右边。

1. 使用多态的好处

无论右边new的时候换那个子类对象，等号左边调用犯法都不会改变。

如：Employee one=new Teacher（）； Employee two=new Assistant();

1. 对象的向上转型

即为多态写法：父类名称 对象名=new 子类名称（）； Animal animal =new Cat();

含义：右侧创建一个子类对象，把它当作父类来看待使用。

注意事项：向上转型一定是安全的，从小范围转向了大范围。

类似于:double num=100;//int🡪double ,正确，自动类型转换。

弊端：对象一旦向上转型为父类，那么就无法调用子类本来特有内容。

弊端的解决方案：用对象的向下转型【还原】。

1. 对象的向下转型

格式：子类名称 对象名=（子类名称）父类对象；

含义:将父类对象，【还原】成为本来的子类对象。

注意事项：

* + 1. 必须保证对象本来创建的时候是“猫”，才能向下转型为“猫”。
    2. 如果对象创建的时候本来不是猫，现在非要向下转型成为猫，就会报错。

（java.lang.ClassCastException,类转换异常）

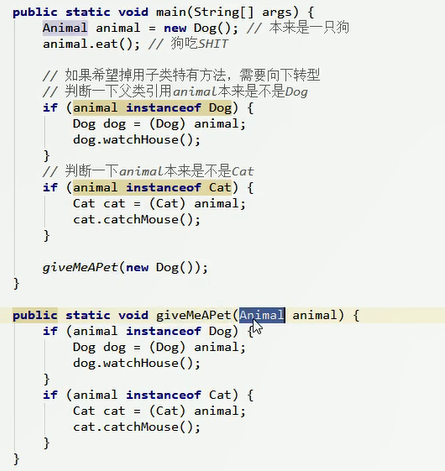
（类似于 int num=(int)10.5;//出错，有精度损失 int num=(int )10.0;//可以

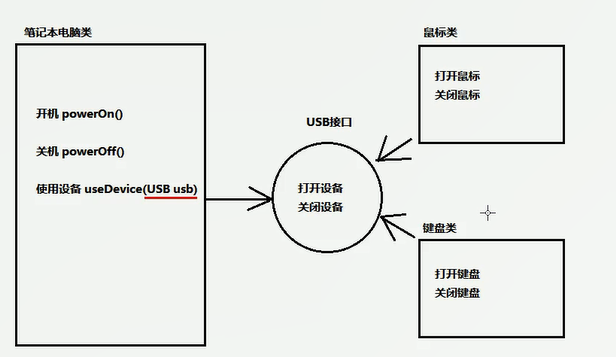
1. 用instanceof关键字进行类型判断

如何知道一个父类引用的对象，原来是什么子类？

格式： 对象 instanceof 类型（这将会得到一个Boolean类型值结果）

举例：if(animal instance of Dog){//…}

深度举例：要求写一个giveMeAPet(Animal animal)方法，使得随机产生一个动物以及打印动物习性。则

1. 笔记本综合案例
2. 案例分析:各个接口和类需要满足的功能，分析类中各个功能与接口是否有实现关系。
3. 注意向上转型、向下转型的使用。（向上转型成usb进行开关设备操作，向下转型来调用不同实现类的特有方法（点击/打印等））

**Day11-【final、权限、内部类】**

【final】

final关键字代表最终、不可改变的。

可以用来修饰一个类、方法、局部变量、成员变量。

1. final关键字修饰一个类，

格式:public final class 类名称{ //…}

含义：当前这个类不能有任何的子类，即为【太监类】

注意：这个类的所有成员方法都无法进行覆盖重写。

1. final关键字修饰一个方法

格式：修饰符 final 返回值类型 方法名称（参数列表）{//方法体}

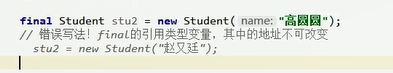
注意：这个方法是最终方法，也就是不能被覆盖重写。

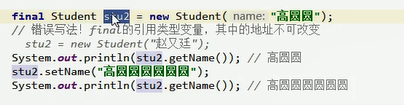
对于类、方法来说，abstract关键字和final关键字不能同时使用，因为矛盾。

1. final关键字作用与局部变量

一旦使用final来修饰局部变量，那么这个变量就不能再进行更改。但是，创建变量时未赋值，再用代码进行赋值也没有错。如:final int num; num=10;//正确。

注意：对于基本类型来说，不可变说的是变量当中的数据不可改变；

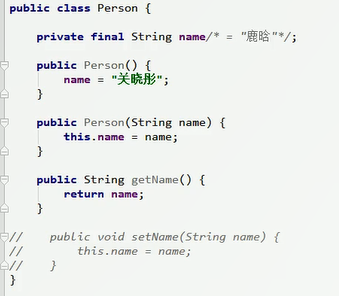
 对于引用类型来说，不可变说的是变量当中的地址值不可改变。

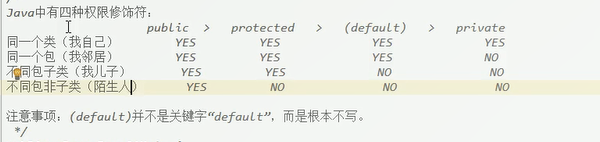
 正确写法（可通过set方法来改变引用变量的值）：

1. final关键字用来修饰成员变量

对于成员变量来说，如果使用final关键字，那么这个变量也照样是不可变。

注意：

1. 由于成员变量具有默认值，所以用了final之后必须手动赋值，不会再给默认值了。
2. 对于final的成员变量，要么使用直接赋值，要么通过构造方法赋值，二者选其一。
3. 必须保证类当中所有重载的构造方法，都最终会对final的成员变量进行赋值。（特别是无参构造函数）

【权限修饰符】

【内部类】

如果一个类内部包含另一个类，那么被包含的类就是内部类。（类似身体和心脏）

分类:①成员内部类 ②局部内部类（包含匿名内部类）

1. 成员内部类的定义格式：

修饰符 class 外部类名称{

修饰符 class 内部类名称{

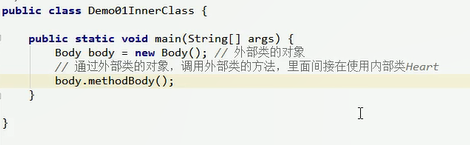
//…}

//…

}

注意：内用外，随意访问；外用内，需要内部类对象。（在body类中写 new Heart().beat();）

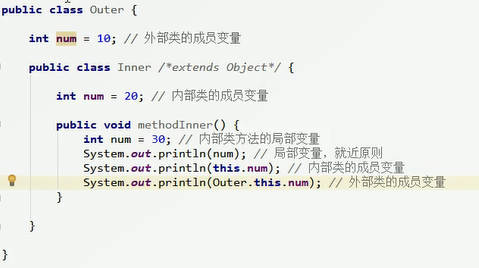
2.如何使用成员内部类？有两种方式：

1. 间接方式：在外部类的方法当中，使用内部类，然后main只是调用外部类方法。

②直接方法， 公式：

类名称 对象名=new 类名称（）；

外部类名称.内部类名称 对象名=new 外部类名称（）.new 内部类名称（）；

3.内部类的同名变量访问

4.局部内部类定义

如果一个类是定义在一个方法内部，那么这就是一个局部内部类。

“局部”：只有在当前所属方法才能使用它，出了这个方法外面就不能使用了。

格式：

修饰符 class 外部类名称{

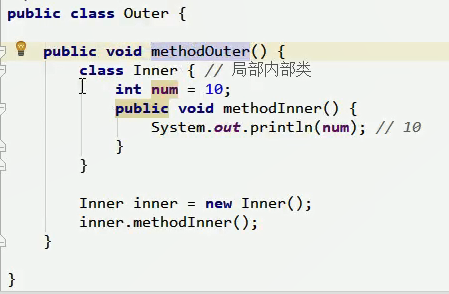
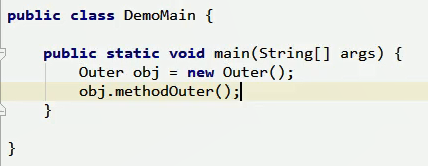
修饰符 返回值类型 外部类方法名称（参数列表）{

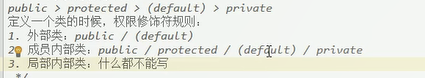
class 局部内部类名称{

//…

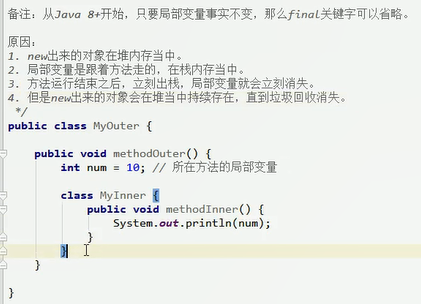
}

}

}

小结以下类的权限修饰符：

1. 局部内部类的final问题局部内部类，如果希望访问所在方法的局部变量，那么这个局部变量必须是【有效final的】。

（局部变量事实不变，即在num=10;之后不再对其值进行赋值改变，则final可以省略）

1. 匿名内部类

如果接口的实现类（或是父类的子类）只需要使用唯一的一次，那么这种情况下就可以省略掉该类的定义，而改为使用【匿名内部类】。

定义格式：接口名称 对象名=new 接口名称（）{ //覆盖重写所有抽象方法}；（分号不要漏）

Tips:大括号及其内容就是匿名内部类，覆盖重写的抽象方法就是在接口中定义的。

解析:

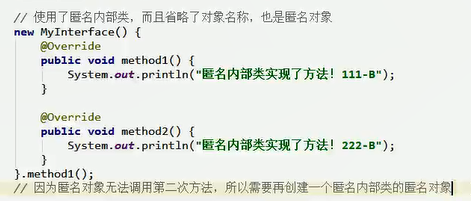
* 1. new代表创建对象的动作 ②接口名称就是匿名内部类需要实现哪个接口
  2. {…}这才是匿名内部类的内容

1. 使用匿名内部类的注意事项
   1. 匿名内部类，在创建对象的时候，只能使用唯一一次。

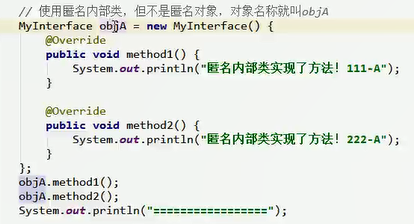
如果希望多次创建对象，并且内容一样的话，那就必须单独定义的实现类了。

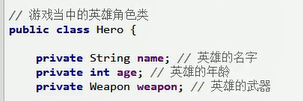
* 1. 匿名对象，在【调用方法】的时候，只能调用唯一一次。

如果希望同一个对象，调用多次方法，那么必须给对象起个名字。

（不能{}.method1().method2();）

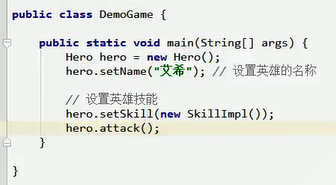
* 1. 匿名内部类是省略了【实现类/子类名称】，但是匿名对象是省略了【对象名称】

强调:匿名内部类和匿名对象不是一回事

8．类作为成员变量类型

图中Weapon类型为自己创建的类。

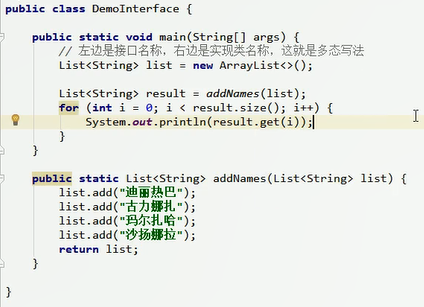
注意：对于 age，name等属性直接写即可，但是weapon是一个类对象，所以需要用到get/set方法得到相应的值（如右图）。

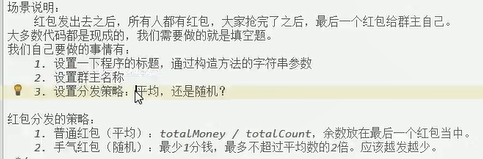
9.接口作为成员变量类型。

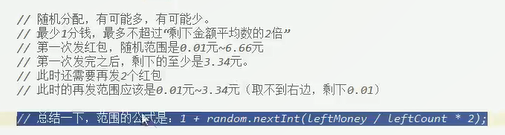
注释:Skill为接口，SkillImpl为该接口的实现类，故设置技能的参数为new SkillImpl()。除了这个方法，还可以使用匿名内部类

进一步简化，同时使用匿名内部类和匿名对象

10.接口作为方法的参数和返回值

小拓展：java.util.list正是ArrayList所实现的接口。

11.发红包综合案例

随机红包分析: