方法中的参数列表叫做形式参数，没有具体的值，只是为了方便在方法中使用

调用方法的时候实际传入的值叫做实际参数，代表具体的数值，用来替换在方法体中代码逻辑的值进行运算

⭐注意：

形式参数的变量名称也是局部变量

当方法的参数值是基本数据类型的时候，不会改变原来的值

当方法的参数值是引用数据类型的时候，如果改变了该引用类型的值，会改变原来的值

总结：java中的参数传递是值传递

**public class** ArgumentDemo {  
  
 **public static void** test(**int** a,**int** b){  
 **int** tmp = a;  
 a = b;  
 b = tmp;  
 }  
  
 **public static void** main(String[] args) {  
 **int** a = 10;  
 **int** b = 20;  
 *test*(a,b);  
 System.***out***.println(a);  
 System.***out***.println(b);  
 }  
}

执行结果：

10

20

a和b的值并没有互换。因为定义的test方法里的a和b是形式参数，在给a和b赋值之后，相当于test方法里的a和b为赋的值，但是因为形式参数是局部变量，因此a和b互换完的结果只是在该方法里有效，而打印的a和b的值还是为赋的值

**public class** Point {  
  
 **private int x**;  
 **private int y**;  
  
 **public** Point(**int** x ,**int** y){  
 **this**.**x** = x;  
 **this**.**y** = y;  
 }  
  
 **public int** getX() {  
 **return x**;  
 }  
  
 **public void** setX(**int** x) {  
 **this**.**x** = x;  
 }  
  
 **public int** getY() {  
 **return y**;  
 }  
  
 **public void** setY(**int** y) {  
 **this**.**y** = y;  
 }  
  
 **public static void** test(Point p){  
 **int** x = p.getX();  
 **int** y = p.getY();  
 **int** tmp = x ; **//1**  
 x = y; **//2**  
 y = tmp; **//3**  
 p.setX(x); **//4**  
 p.setY(y); **//5**

**//1~5行也可以写为：**

**p.setX(y);**  
**p.setY(x);**

}  
}

**public class** ArgumentDemo {  
  
 **public static void** main(String[] args) {  
 Point p1 = **new** Point(2,3);  
 Point.*test*(p1);  
 System.***out***.println(p1.getX());  
 System.***out***.println(p1.getY());  
 }  
}

执行结果：

3

2

**封装**

* 为什么要使用封装？

下面代码有什么缺陷？

Dog d = new Dog();

d.health = -1000; 健康值是不能为负数的

再举一个例子：

**public class** Dog {  
  
 String **name**;  
 **int age**;  
 String **color**;  
  
 **public** Dog(){  
 }  
  
 **public** Dog(String name,**int** age,String color){  
 **this**.**name** = name;  
 **this**.**age** = age;  
 **this**.**color** = color;  
 }  
 **public void** eat(){  
 System.***out***.println(**"eating bones"**);  
 }  
  
 **public void** play(){  
 System.***out***.println(**"playing ..."**);  
 }  
  
 **public void** show(){  
 System.***out***.println(**"name:"**+**this**.**name**);  
 System.***out***.println(**"age:"**+**this**.**age**);  
 System.***out***.println(**"color:"**+**this**.**color**);  
 }  
 **public static void** main(String[] args) {  
 Dog dog = **new** Dog();  
 dog.**name** = **"大黄"**;  
 dog.**age** = 20;  
 dog.**color** = **"yellow"** ;  
 dog.show();  
 }

}

这时可以实现打印出来想要的结果，但是即使我将dog.age的值赋与负数也不会报错，依然会显示。如何在赋值的同时添加一些逻辑判断呢？使用封装可以解决此问题

可以使用封装，保证数据的规范。不符合规范的数据将无法进行操作

刚刚的代码添加如下代码：

**public void** setAge(**int** age){  
 **if**(age>0){  
 **this**.**age** = age;  
 }**else**{  
 System.***out***.println(**"输入年龄不规范，请重新输入"**);  
 }  
}

这时dog.setAge(-20);就会显示输入年龄不规范，请重新输入

但是这是调用了这个方法，而之前的dog.age=-20的时候依然不会报错，还是会正常显示。

这时如果想让其直接就报错可以这样写↓

定义的成员变量**int age**;写成这样 → **private int age**;

然后将main方法里的代码写成这样↓

Dog dog = **new** Dog();  
dog.**name** = **"大黄"**;  
dog.setAge(-20);  
dog.**color** = **"yellow"** ;  
dog.show();

执行结果：

输入年龄不规范，请重新输入

name:大黄

age:0 //因为输入年龄无效，因此显示int类型的默认值0

color:yellow

* 面向对象的封装的概念：

将类的某些信息隐藏在类内部，不允许外部程序直接访问，而是通过该类提供的方法来实现堆隐藏信息的操作和访问。即将类中的属性设置为私有属性，提供共有的外部方法供程序进行调用，可以实现丰富的细节操作

* 广义的封装的概念：

可以将完成特定功能的代码块封装成一个方法，供不同的程序进行调用

* 封装的好处：

-只能通过规定方法访问数据

-隐藏类的实现细节

-方便加入控制语句（逻辑判断语句）

-方便修改实现

为什么需要封装？封装的作用和含义？

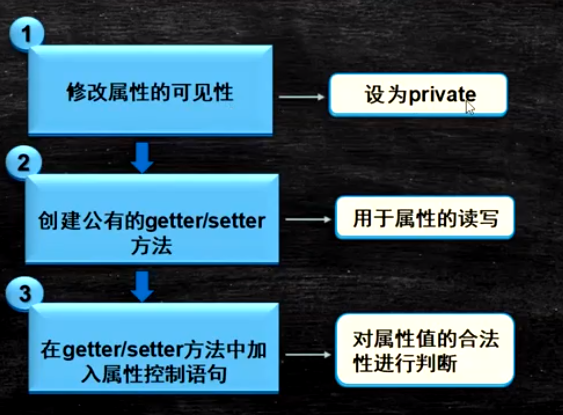
我要看电视，只需要按一下开关和换台就可以了，没有必要了解电视机内部的构造。

即封装就是隐藏对象内部的复杂性，只对外公开简单的接口，便于外界调用，从而提高系统的可扩展性，可维护性。

我们程序设计要追求“高内聚，低耦合”。高内聚：类的内部数据操作细节自己完成，不允许外部干涉；低耦合：仅暴露少量的方法给外部使用

* 如何使用封装：

-封装的步骤：



即里面有多少个成员属性，就写多少对get,set方法

定义类的时候需要包含以下组件：

-私有属性

-构造方法（无参构造方法和自定义构造方法）

-set get方法

-普通方法

上面Dog类的代码可以用封装的方法写成↓这样：

在Dog.java里：

**public class** Dog {  
  
 **private** String **name**;  
 **private int age**;  
 **private** String **color**;  
  
 **public** Dog(){  
 }

**public void** setAge(**int** age){  
 **if**(age>0){  
 **this**.**age** = age;  
 }**else**{  
 System.***out***.println(**"输入年龄不规范，请重新输入"**);  
 }  
 }  
  
 **public int** getAge(){  
 **return this**.**age**;  
 }  
  
 **public void** setName(String name){  
 **this**.**name** = name;  
 }  
  
 **public** String getName(){  
 **return this**.**name**;  
 }  
  
 **public void** setColor(String color) {  
 **this**.**color** = color;  
 }  
  
 **public** String getColor(){  
 **return this**.**color**;  
 }

}

在DogTest里：

**public class** DogTest {  
 **public static void** main(String[] args) {  
 Dog dog = **new** Dog();  
 dog.setName(**"大黄"**);  
 dog.setAge(20);  
 dog.setColor(**"yellow"**);  
 System.***out***.println(dog.getAge());  
 System.***out***.println(dog.getName());  
 System.***out***.println(dog.getColor());  
 }  
}

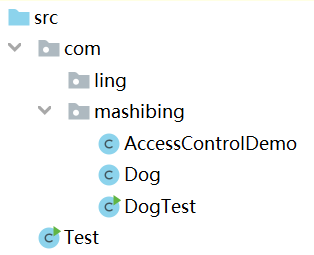
**面向对象的三大特征：**

* 封装/隐藏 encapsulation

-对外隐藏某些属性和方法

-对外公开某些属性和方法

-使用访问控制符实现封装（作用是限制访问）



**成员(成员变量或成员方法)访问权限共有4种**：

💗例子中AccessControlDemo类和Test类属于不同包。

AccessControlDemo属于com.mashibing包，而Test和com是一个层级的

1. **public** 公共的

可以被当前项目中所有的类访问（项目可见性）

AccessControlDemo包中：

**public class** AccessControlDemo {

**public** String **str** = **"test"**;

}

Test包中:

**public class** Test {  
**public static void** main(String[] args) {  
 AccessControlDemo acd = **new** AccessControlDemo();  
 System.***out***.println(acd.**str**);

}

}

执行结果：test

1. **protected** 受保护的

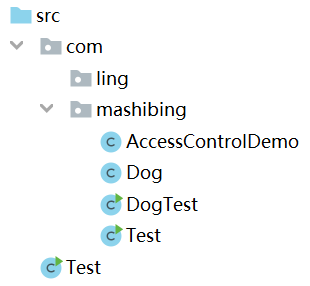
可以被这个类本身访问；

同一个包中的所有其他的类访问；

被它的子类（同一个包以及不同包中的子类）访问；

将AccessControlDemo包中的**public** String **str** = **"test"**;改成

**private** String **str** = **"test"**;之后再去Test包里发现执行报错



但是这时我在com.mashibing包里建一个Test，

代码和在Test中的代码一样，发现可以执行打印出test

1. **default / friendly** 默认的/友好的（包可见性）

被这个类本身访问；

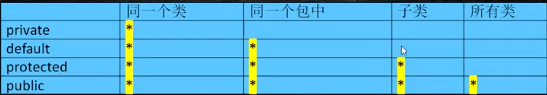
被同一个包中的类访问；

和protected的前两项是一个道理，这里例子略

1. **private** 私有的

只能被这个类本身访问（类可见性）

例子中即在不同同一个包，在同一个包都不能访问，只能在当前类中访问



**类的访问权限只有两种：**

-public 公共的

可被同一项目中所有的类访问（必须与文件名同名）

-default/friendly 默认的/友好的

可被同一个包中的类访问

封装要点：

类的属性的处理：

一般使用private，除非本属性确定会让子类继承

提供相应的get/set方法来访问相关属性，这些方法通常是public，从而提供对属性的读取操作（注意：boolean变量的get方法是用 **:is**开头）

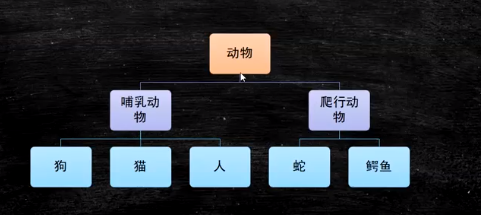
一些只用于本类的辅助性方法可以用private，希望其他类调用的方法用public

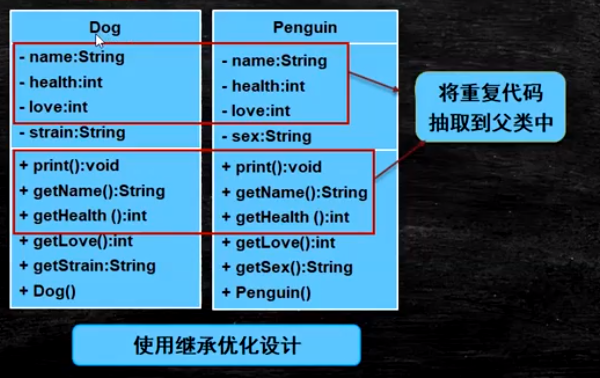
* 继承inheritance

-子类 父类

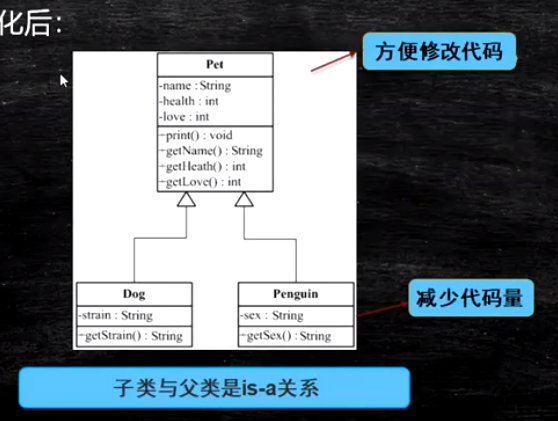
-子类可以从父类继承属性和方法

-子类可以提供自己单独的属性和方法





使用继承优化后：



-概念：

表示父类跟子类之间的关系，当两个类或者多个类具备相同的属性和方法的时候可以提取出来变成父类(超类)，子类(基类/派生类)可以继承。子类和父类的关系是 is-a

类是对对象的抽象，继承是对某一批类的抽象，从而实现对现实世界更好的建模

-使用：

使用继承的时候需要使用extends关键字。子类是父类的扩展

使用继承关系之后父类中的属性和方法都可以在子类中进行使用(非私有属性和非私有方法)

java中是**单继承**关系，即一个子类不能同时继承两个或两个以上的父类。因为如果包含多个父类，同时父类中包含重名方法，无法决定该调用谁

例子：

现在有Dog和Penguin两种动物都属于Pet，并且Dog和Penguin有相同的属性和方法，为name，age，gender。Dog有独有的属性sound；Penguin有独有的属性color。

原本的代码应该是↓

Dog.java:

**public class** Dog{  
  
 **private** String **name**;  
 **private int age**;  
 **private** String **gender**;  
 **private** String **sound**;  
  
 **public** Dog(){  
 }  
  
 **public** Dog(String name,**int** age,String gender,String sound){  
 **this**.**name** = name;  
 **this**.**age** = age;  
 **this**.**gender** = gender; **this**.**sound** = sound;  
 }  
  
 **public void** setName(String name){  
 **this**.**name** = name;  
 }  
  
 **public** String getName(){  
 **return name**;  
 }  
  
 **public void** setAge(**int** age){  
 **this**.**age** = age;  
 }  
  
 **public int** getAge(){  
 **return age**;  
 }  
  
 **public void** setGender(String gender){  
 **this**.**gender** = gender;  
 }  
  
 **public** String getGender(){  
 **return gender**;  
 }  
  
 **public void** setSound(String sound){  
 **this**.**sound** = sound;  
 }  
  
 **public** String getSound(){  
 **return sound**;  
 }  
}

Penguin.java里：

**public class** Penguin {  
  
 **private** String **name**;  
 **private int age**;  
 **private** String **gender**;  
 **private** String **color**;  
  
 **public** Penguin(){  
  
 }  
  
 **public** Penguin(String name,**int** age,String gender,String color){  
 **this**.**name** = name;  
 **this**.**age** = age;  
 **this**.**gender** = gender;  
**// super(name,age,gender);  
 this**.**color** = color;  
 }  
  
 **public void** setName(String name){  
 **this**.**name** = name;  
 }  
  
 **public** String getName(){  
 **return name**;  
 }  
  
 **public void** setAge(**int** age){  
 **this**.**age** = age;  
 }  
  
 **public int** getAge(){  
 **return age**;  
 }  
  
 **public void** setGender(String gender){  
 **this**.**gender** = gender;  
 }  
  
 **public** String getGender(){  
 **return gender**;  
 }  
  
 **public void** setColor(String color){  
 **this**.**color** = color;  
 }  
  
 **public** String getColor(){  
 **return color**;  
 }  
}

但是因为它们有重合的属性和方法，因此把它们归拢到父类中，这里定义为Pet类

Pet.java中：

**public class** Pet {  
  
 **private** String **name**;  
 **private int age**;  
 **private** String **gender**;  
  
 **public** Pet(){  
  
 }  
  
 **public** Pet(String name,**int** age,String gender){  
 **this**.**name** = name;  
 **this**.**age** = age;  
 **this**.**gender** = gender;  
 }  
  
 **public void** setName(String name){  
 **this**.**name** = name;  
 }  
  
 **public** String getName(){  
 **return name**;  
 }  
  
 **public void** setAge(**int** age){  
 **this**.**age** = age;  
 }  
  
 **public int** getAge(){  
 **return age**;  
 }  
  
 **public void** setGender(String gender){  
 **this**.**gender** = gender;  
 }  
  
 **public** String getGender(){  
 **return gender**;  
 }  
  
}

dog.java中: （注释的部分都是不要的）

**public class** Dog **extends** Pet{  
  
**// private String name;  
// private int age;  
// private String gender;  
 private** String **sound**;  
  
 **public** Dog(){  
  
 }  
  
 **public** Dog(String name,**int** age,String gender,String sound){  
**// this.name = name;  
// this.age = age;  
// this.gender = gender;  
 super**(name,age,gender);  
 **this**.**sound** = sound;  
 }  
**//  
// public void setName(String name){  
// this.name = name;  
// }  
//  
// public String getName(){  
// return name;  
// }  
//  
// public void setAge(int age){  
// this.age = age;  
// }  
//  
// public int getAge(){  
// return age;  
// }  
//  
// public void setGender(String gender){  
// this.gender = gender;  
// }  
//  
// public String getGender(){  
// return gender;  
// }  
  
 public void** setSound(String sound){  
 **this**.**sound** = sound;  
 }  
  
 **public** String getSound(){  
 **return sound**;  
 }  
}

Penguin.java中: （注释的部分都是不要的）

**public class** Penguin **extends** Pet{  
  
**// private String name;  
// private int age;  
// private String gender;  
 private** String **color**;  
  
 **public** Penguin(){  
  
 }  
  
 **public** Penguin(String name,**int** age,String gender,String color){  
**// this.name = name;  
// this.age = age;  
// this.gender = gender;  
 super**(name,age,gender);  
 **this**.**color** = color;  
 }  
  
**// public void setName(String name){  
// this.name = name;  
// }  
//  
// public String getName(){  
// return name;  
// }  
//  
// public void setAge(int age){  
// this.age = age;  
// }  
//  
// public int getAge(){  
// return age;  
// }  
//  
// public void setGender(String gender){  
// this.gender = gender;  
// }  
//  
// public String getGender(){  
// return gender;  
// }  
  
 public void** setColor(String color){  
 **this**.**color** = color;  
 }  
  
 **public** String getColor(){  
 **return color**;  
 }  
}

再定义一个PetTest类里面写main函数：

**public class** PetTest {  
 **public static void** main(String[] args) {  
 Dog dog = **new** Dog();  
 dog.setName(**"大黄"**);  
 System.***out***.println(dog.getName());  
 }  
}

执行结果：

大黄

setName在Dog类（子类）中没有这个方法，在Pet类（父类）中有，因为通过继承子类继承了父类的属性和方法，因此可以调用该方法。但是如果在Pet类中定义一个private的方法，在PetTest里想通过dog.该private方法名 是无法调用的。因为即使继承也是无法继承私有属性和私有方法的

**super关键字**

super是直接父类对象的引用

比如现在A是B的父类，B是C的父类，C是D的父类，那么在D里写super()调用的是C里的对象，在C里写super()调用的是B里的对象，在B里写super()调用的是A里的对象

可以通过super来访问父类中被子类覆盖的方法或属性。

格式：super**.**父类方法名称

super不能出现在static方法中

比如刚才的Dog.java里定义

**public void** play(){  
 System.***out***.println(**"dog is playing the ball"**);  
}

在Pet.java里定义

**public void** play(){  
 System.***out***.println(**"play"**);  
}

这时在PetTest.java里

**public class** PetTest {  
 **public static void** main(String[] args) {  
 Dog dog = **new** Dog();  
 dog.play();  
 }  
}

执行结果为：**dog is playing the ball**

如果想让它也调Pet里的play方法，就需要使用super了，在Dog.java里：

**public void** play(){  
 **super**.play(); **//加上←这句话**  
 System.***out***.println(**"dog is playing the ball"**);  
}

这时在主方法里的执行结果为：

**play**

**dog is playing the ball**

普通方法： 没有顺序限制，可以随便调用

构造函数中：当super在构造方法中使用时，会调用父类的构造方法，一定要将super放在

第一行

在构造方法中，super关键字和this关键字(这里指的是方法，不是属性)不能同时出现

比如上面代码的↓

**public** Penguin(String name,**int** age,String gender,String color){  
 **super**(name,age,gender);  
 **this**.**color** = color;  
}  
这里如果再加个this();就会报错，因为它是this方法，而this.color是属性，没关系

子类的构造方法中都会默认使用super关键字调用父类的构造方法。因此在定义类的时候，无论自己是否定义了其他构造方法，最好将无参构造方法写上

💗💗💗💗💗💗💗💗💗💗💗💗💗💗💗💗💗💗

比如在Dog.java中将无参构造方法写成↓

**public** Dog(){  
 System.***out***.println(**"dog无参方法"**);  
}

在Pet.java中将无参构造方法写成↓

**public** Pet(){  
 System.***out***.println(**"pet无参构造方法"**);  
}

在PetTest.java的main方法里new

Dog dog = **new** Dog();

💗💗💗💗💗💗💗💗💗💗💗💗💗💗💗💗💗💗

执行结果：

pet无参构造方法

dog无参方法

💗在保持上面代码的基础之上进行下面的举例：💗

如果我只是在Dog.java中将无参构造方法写成↓

**public** Dog(){  
 System.***out***.println(**"dog无参方法"**);  
}

将Pet.java中↓代码注释上会报错

**//public** Pet(){  
**//** System.***out***.println(**"pet无参构造方法"**);  
**//}**

因为在调用构造方法Dog()的时候，它里面的第一行是会自动给加上super();进行调用。这时就要求父类里面必须有一个无参的构造方法，不然就会报错

如果我在PetTest.java里new的时候new的是带参数的构造方法对象

Dog dog = **new** Dog(**"小黑"**,12,**"男"**,**"汪汪"**);

这时无参的构造方法的内容就都不会被打印出来

如果在pet.java里

**public** Pet(String name,**int** age,String gender){  
 **this**(); **//新加←代码**  
 **this**.**name** = name;  
 **this**.**age** = age;  
 **this**.**gender** = gender;  
}

这时在main方法里执行↓

Dog dog = **new** Dog(**"小黑"**,12,**"男"**,**"汪汪"**);

就会打印出**pet无参构造方法**

如果构造方法中显示的指定了super的构造方法，那么无参的构造方法就不会被调用。如果在有参的构造方法中没有指定super的构造方法，那么无参的构造方法就会被调用

如果将Dog.java里的有参构造方法里的↓代码注释上

**super**(name,age,gender);

并且刚才的例子中的this();删掉

也会打印**pet无参构造方法**

⭐总结：

在创建子类对象的时候一定会优先创建父类对象

所有的java类都具备同一个老祖宗类，称之为java.lang.Object，是所有类的根类

**理解继承**

子类访问父类成员。使用super关键字，super代表父类对象

-访问父类构造方法

super();

super(name);

-访问父类属性

super.name;

-访问父类方法

super.print();

⭐**不能被继承的父类成员**

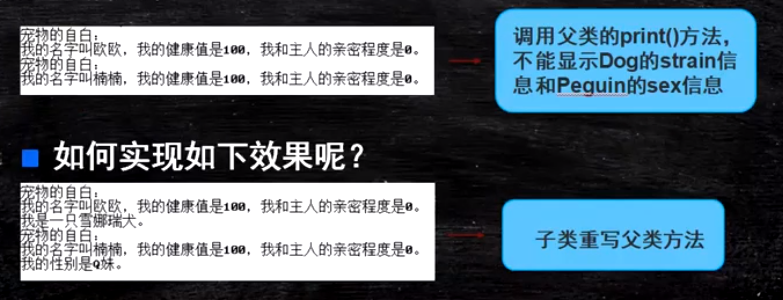
不能直接访问private成员

子类与父类不在同包，使用默认访问权限的成员

构造方法

**方法重写**

使用继承后效果



**重写**

-表示的是子类覆盖父类的方法，当覆盖之后，调用同样的方法的时候会优先调用子类

-必须要存在继承关系，当父类中的方法无法满足子类需求的时候可以选择重写的方式

-当子类里面出现了和父类里面同名的方法，相同的参数列表和返回类型的时候（即只有方法体不一样，其他的必须一模一样），在子类的这个同名的方法上面写上 **@Override** 就会实现重写。

-重写方法不能使用比重写方法更严格的访问权限。比如父类里的方法是public的，而重写的方法是protected就不行，即只要是比public权限严格的都不行

-父类的静态方法子类可以进行调用，但是子类不可以重写

在PetTest里写

**public class** PetTest {  
 **public static void** main(String[] args) {Pet pet = **new** Pet();  
 System.***out***.println(pet);  
 }  
}

执行结果：

pet无参构造方法

com.mashibing.Extend.Pet@b4c966a //这里@及后面的加在一起表示内存地址

如果想要将具体的值打印出来的话，要在Pet这个类里面添加一个toString()的方法再打印

在Pet.java里：

@Override  
**public** String toString(){  
 **return "my name is "**+**this**.**name**+**",my age is "**+**this**.**age**+**",my gender is "**+**this**.**gender**;  
}

这样执行main方法时打印出来结果为：

pet无参构造方法

my name is null,my age is 0,my gender is null

这时我在Dog.java里想要让new出来的dog除了有Pet里定义的那些属性以外，还想要属于它自己的属性sound的时候：

Dog.java里：

**public** String toString(){  
 **return super**.toString()+**", my sound is "**+**this**.**sound**;  
}

main方法里：

Dog dog = **new** Dog();  
System.***out***.println(dog);

执行结果：

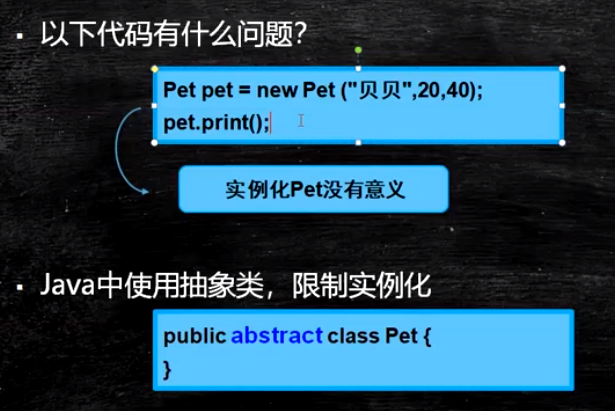
pet无参构造方法

dog无参方法

my name is null,my age is 0,my gender is null, my sound is null



**抽象类**



没有一种宠物叫宠物（猫，狗是宠物，可以new出猫/狗 Dog dog = new Dog(“贝贝”，20)），没有一种人叫人（男人，女人是人，可以new出男人/女人）

java中的对象是对现实世界的具象化，但是在现实世界中，某些类并不具备实例化的意义。因此可以定义为抽象类

**注意：**

-创建抽象类的时候需要添加abstract关键字

-不能进行实例化（也就是不能new对象）

-抽象类中的某些方法需要子类进行更丰富的实现，父类实现没有意义，此时可以将抽象类

中的方法定义为抽象方法，没有具体的实现，只包含方法名称，返回值，参数列表，访问

修饰符

-使用abstract关键字修饰的方法叫做抽象方法，可以不写方法的实现

-有抽象方法的一定是抽象类，但是抽象类里不一定有抽象方法

比如我现在有一个类叫Pet，我在里面定义：

**public abstract class** Pet {  
 **public void** print(){  
 }  
}

在同一个包里定义一个类AbstratTest：

**public class** AbstractTest {  
 Pet **pet** = **new** Pet();  
}

红色部分报错，因为在Pet里面已经定义这个类为抽象类，无法进行实例化，因此报错

比如这个抽象类的print()没有什么实际意义，就可以不写方法体

**public abstract class** Pet {  
  
 **private** String **name**;  
 **private int age**;  
  
 **public abstract void** print(); //该方法没有{}括起来的方法体  
}

⭐子类在继承父类时，必须要将父类中的抽象方法进行实现或者将子类也定义为抽象类

上面的抽象类写完以后，比如现在有个子类为Dog，里面需要实现print（）这个方法：

**public class** Dog **extends** Pet{  
  
 **private** String **gender**;  
  
 @Override  
 **public void** print(){  
 System.***out***.println(**"dog print"**);  
 }  
  
}

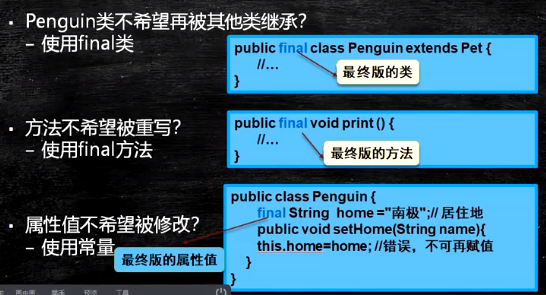
这样我在main方法里：

Dog dog = **new** Dog();  
dog.print();

执行结果：

dog print

**final用法**



-final可以修饰变量，表示变量的值不可变

**public static final int *age*** = 10;  
**public static void** main(String[] args) {  
 ***age*** = 20;  
}

-final可以修饰方法

**public class** FinalDemo {

**public final void** test(){  
 System.***out***.println(**"final test"**);  
}

}

在和它同样的包里：

**public class** FinalSubClass **extends** FinalDemo{  
 @Override  
 **public final void** test(){  
 }  
}

这样会报错，因为父类里的方法定义的是final方法，无法进行重写

-final可以修饰类，表示类不可以被继承

比如我在父类里

**public final class** FinalDemo {

}

在子类里

**public class** FinalSubClass **extends** FinalDemo{

}

会报错

* 多态 polymorphism

-为了适应需求的多种变化，使代码变得更加统用

※面向过程只有封装性（功能的封装，而没有数据的封装），没有继承和多态