**面向对象编程**

一个问题？[Demo107.java]

张老太养了两只猫猫：一只名字叫小白，今年3岁，白色。还有一只叫小花，今年5岁，花色。请编写一个程序，当用户输入小猫的名字时，就显示该猫的名字，年龄，颜色。如果用户输入的小猫名错误，则显示张老太没有这只猫猫。

跟据以往学习的知识，解决问题：

语言的发展从低级到高级

汇编语言(面向机器)-----c语言(基于对象/面向过程)-------java(面向对象)---------？

那么Java中如何实现面向对象的呢

**类的概念：**

**类是什么，把一类事物相同的属性提取出来封装成一个文件，这个文件就叫类（class）**

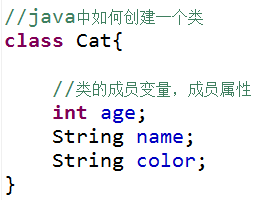
**对象的概念(对象实例)**

**对象就是类的具体的一个实现，狗类下的旺财**

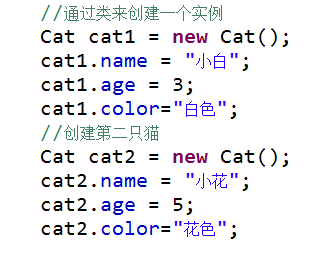
**类是某种属性的集合，对象就是集合中的具体的个例**

案例：在刚才的代码中改进，使用面向对象的方法来解决我们的问题

**1、首先创建一个猫类(类名要大写)**



**2、通过猫类来创建一个对象实例**



//通过该案例，我们学习以下知识点

**1、类是抽象的概念的，是一类事物的共同属性、行为的集合**

**2、对象是具体的，实际的，代表一个具体的个例**

**3、在java中类是对象的模版，可通过类来创建具体的对象**

**4、如何创建一个简单的类(不完善)**

**5、创建一个对象的方法**

**1、直接创建**

**Cat cat1 = new Cat()**

**2、先定义再创建**

**Cat cat1 ;**

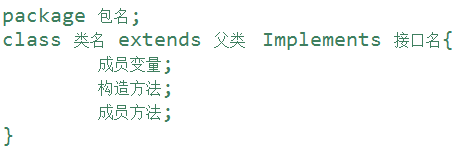
**cat1 = new Cat();**

**5、什么叫类的成员属性/成员变量**

**6、类的属性怎么设置、怎么调用** （对象名.属性名）

这个类很简单，我们看看如何完整的定义一个类

一个完整的类如下图：



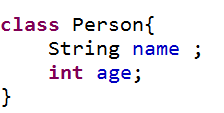
为了使同学们容易接受，我们来一步步讲解

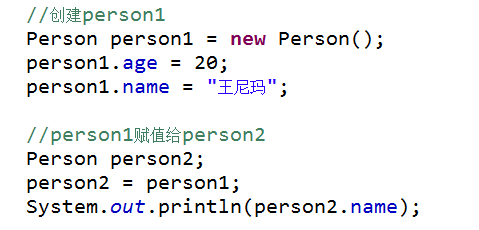
1. **类的成员变量**

成员变量是类的一个组成部分，一般是基本数据类型和引用类型。

比如：猫有一个主人，主人就是一个引用类型

引用类型与基本类型的区别：



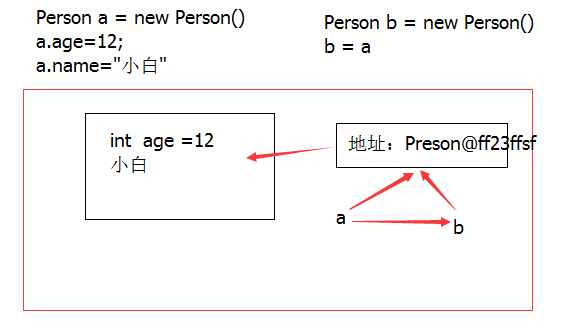


注：可通过

System.out.println(persion1)

打出person1的引用地址

引用类型内存示意图：



1. **成员方法**

成员方法：类是某种属性和行为的集合，我们现在理解了什么是成员属性，下面我们看看什么是成员方法

行为在Java中表现形式就是方法

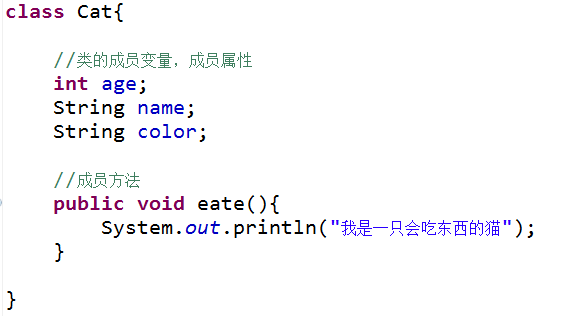
方法形式如下：

public 返回数据类型 方法名(参数列表){

方法体（语句）;

}

Eg:猫都会吃东西、猫都会叫 这些行为表现在java就是方法



成员方法如何调用呢

对象名.方法名();

cat1.eate();

注：方法名的命名规范

方法名一般以小写字母开始，

可以使用 1、驼峰法：catEat

2、下划线法：cat\_eat

现在要让小猫会计算

1. 计算从1加到1000的结果

**public** **void** jiShuan(){

**int** tem = 0;

**for**(**int** i=0;i<1001;i++){

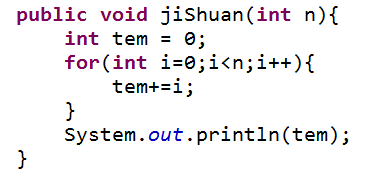
tem+=i;

}

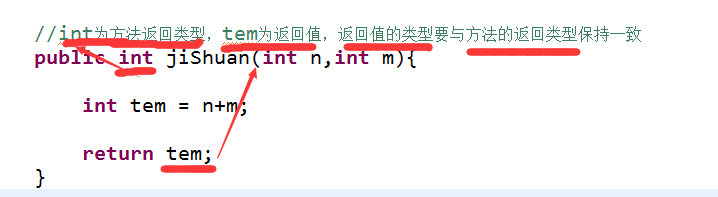
System.*out*.println(tem);

}

1. 计算从1加到指定的数字n的结果



1. 计算两个指定数字相加的结果，并返回该结果供调用对象使用



通过该案例学习：

1. 成员方法的结构，使用

public 返回数据类型 方法名(参数列表){

方法体（语句）;

}

使用：对象名.方法名(参数一，参数二…)

1. 参数的传递，参数的类型
   * 1. 参数的个数可以是0个或多个
     2. 参数的类型要与传递的值的类型一致
2. 方法的返回类型，返回值
   * 1. 方法可以无返回void修饰
     2. 方法的返回类型要与返回值的类型一致

**3、构造方法：**

我们思考这么一个问题，前面我们在创建一个猫类对象的时候，是先创建一个猫类对象，再给这个对象赋初始值，那我们能不能在对象创建的时候就以指定的值来创建呢，这就要用到构造方法了，构造方法的作用是完成新对象的初始化

构造方法特点：

1. 构造方法的方法名和类(public)名相同
2. 没有返回值（连void也没有）
3. 在创建一个类的新对象时，系统会自动调用该类的构造方法完成初始化
4. 每个类都有构造函数，系统会默认给一个无参构造方法，如果有自定义构造方法，则无参方法不起作用。
5. 一个类可以有多个构造函数(方法名一致，参数可以没有或一个或多个)

Eg:

public Cat(){ }

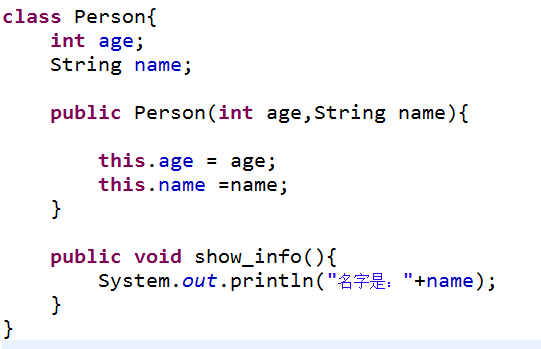
public Cat(int age ,String name){ }

this关键字

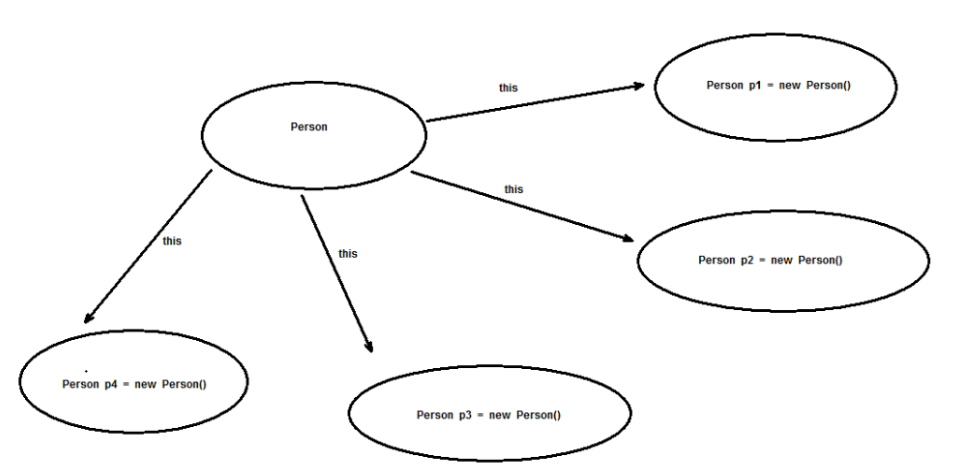
this 用在类中，代表当前创建的对象

java虚拟机会给每个对象分配this代表当前对象，this只能在类中使用，不能到外部使用

使用：



图解：

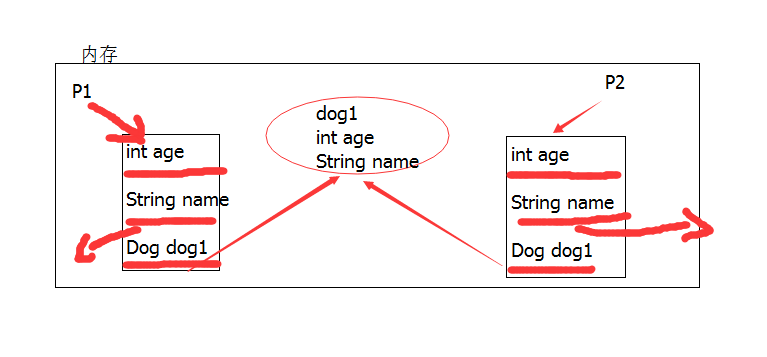


引用数据类型的使用(图解):

Dog dog1 = new Dog(age , name);

Person p1 = new Person(dog1, age, name);

Person p2 = new Person(dog1, age, name);



基本数据类型，在内存中直接开辟空间，引用数据类型，通过地址指向被引用对象

**3.1静态变量**和**静态方法**(类变量和类方法)

**静态变量：**

我们思考这么一个问题：

我们班现在有3个同学，过段时间加入一个，经过一段时间后，我要统计现在班上有多少个学生

传统方法：

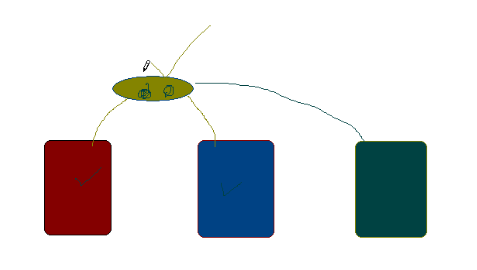
Int total = 0 ;

Stu s1 = new Stu ();

Total++;

面向对象：

设计一个int total表示总人数，加入一个学生的时候，就把total加1，并且total是所有学生共享的就好了。



这种变量，叫静态变量

总结：静态变量是该类所有对象共享的变量，任何一个该类的对象去访问它时，取到的都是相同的值，同样任何一个该类的对象去修改它时，修改的也是同一个变量。

使用：

1. 如何定义静态变量

访问修饰符 static 数据类型 变量名

1. 如何访问静态变量

类名.静态变量名(有权限限制) 或者 对象名. 静态变量名

**静态方法：**

同静态变量一样，所有实例都可以访问到的方法叫静态方法，

如何定义一个静态方法：

访问修饰符 static 返回类型 方法名（）{ 方法体 }

如何访问静态方法

类名.静态方法名(有权限限制) 或者 对象名.静态方法名

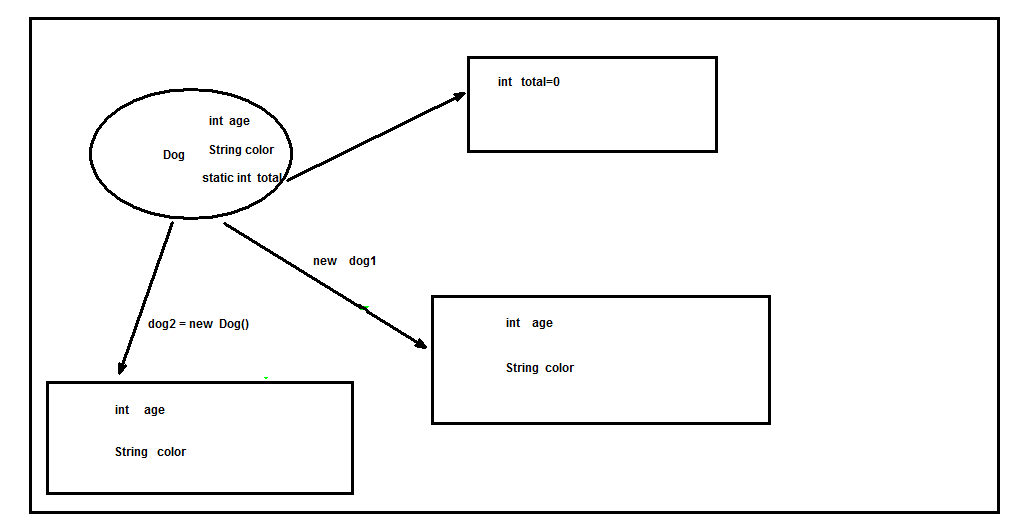
静态方法和属性和成员方法和属性对比总结：

1、最大的区别在于内存。静态方法在程序开始时生成内存,实例方法在程序运行中生成内存，所以静态方法可以直接调用,实例方法要先成生实例,通过实例调用方法，静态速度很快，但是多了会占内存。

  2、静态成员属于类所有，非静态成员属于类的实例/对象所有。

　　3、每创建一个类的实例，都会在内存中为非静态成员新分配一块存储；

4、静态成员属于类所有，为各个类的实例所公用，无论类创建了多少实例，类的静态成员在内存中只占同一块区域



Java面向对象编程的三个基本特性

一、封装 二、继承 三、多态

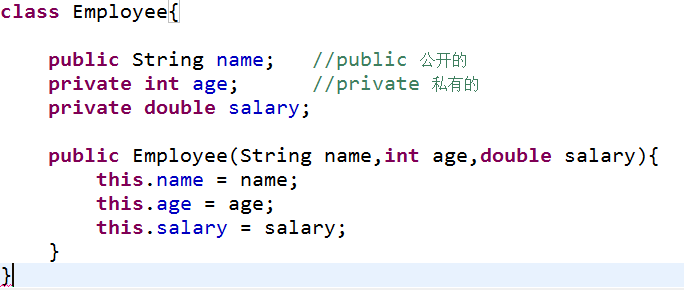
一、封装

封装就是把类的属性和操作封装在一起，数据被保护到内部，程序的其它部分只有通过被授权的操作(成员方法)，才能对数据进行操作

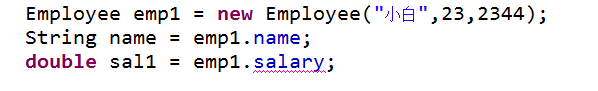
封装是通过访问修饰符来控制的

访问修饰符有四种：public protected 无 private

Eg：定义一个员工类，员工的年龄，工资信息不能随便访问

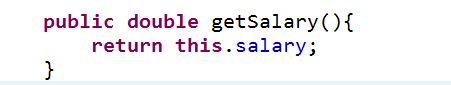


这时就不能通过 对象.属性来该问该属性



那我们怎么在外部访问呢：

在职员类中定义一个公开的方法来获取薪水



现在我们知道了控制修饰符的作用，我们来看一下Java给我们提供了哪些控制修饰符

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，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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 访问级别 | 修饰符 | 同类 | 同包 | 子类 | 不同包 |
| 公开 | Public | *✓* | *✓* | *✓* | *✓* |
| 受保护 | protected | ✓ | ✓ | ✓ | **X** |
| 默认 | 无 | ✓ | ✓ | **X** | **X** |
| 私有 | private | ✓ | **X** | **X** | **X** |

四种修饰符的访问范围

讲解修饰符的范围涉及到一个概念叫包，那什么是包呢，它有什么必要性呢。

我们看下面这个案例

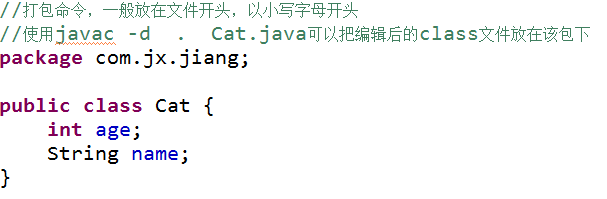
Demo1.java 下有个 Dog类

Demo2.java 下也有个 Dog类

就会产生冲突，我们在开发的过程中呢，会有很多员工同时开发一个项目，如果放在同一个文件夹下，就会产生冲突，解决这个问题就要用到包，每个员工开发的模块放到不同的包下面，这样即使类名相同，也不会发生冲突

1、包的使用方法：

1.1打包 命令 package

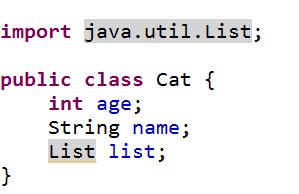


一般我们为了避免使用一些常见的单词做为包名而引起冲突，规定使用公司域名倒过来写作为包名

如:com.sina. com.baidu. 等

1.2 导入包的命令 import

为什么需要导入包，因为有时我们需要用到别人开发的一些类和方法的时候，就需要引入包，比如我们需要用到一个List的类，就需要引入java.util包



使用方法：

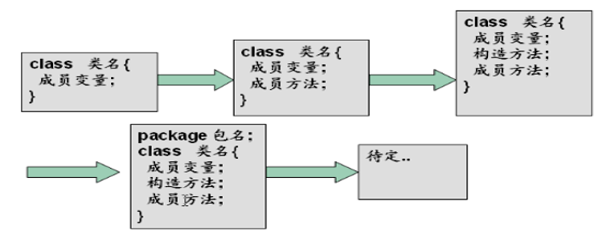
2、包的一些概念

一个包下可以含有很多个类，除了自定义包，java也给我们提供了很多包，可以看下jre下的lib --- rt.jar中含有许多包，我们先看一些常用的包

java.lang.\* 包 自动引入 java.net.\* 包 网络开发包

java.util.\* 包 工具包 java.io.\* 包 文件流包

学习完以上知识后我们的类就更加完善了

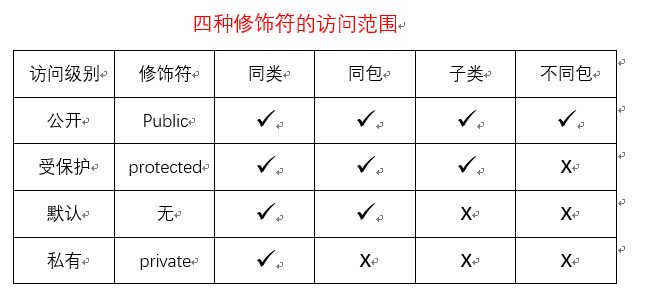


我们现在对包的概有了了解 ，那么包有哪些功能呢，从上面我们可以知道包可以 1、区分相同名字的类

2、当类很多时可以方便我们管理

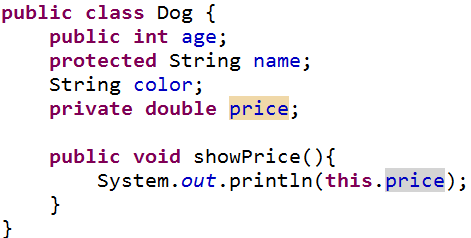
3、控制访问范围

前二个功能我们都很容易理解，那么用包来控制访问范围是怎么实现的呢

c

同类 即在该类下访问

默认的 子类无法访问前提是子类与父类不在同一个包下，



二、继承

我们看这么一个案例

我们养了三只宠物，一只猫，一只狗，一只兔子，使用我们面向对向的方法呢，我们就可以定义三个类，一个猫类，一个狗类，一个兔子类

我们代码如下：

**class** Cat{

**int** age;

String name;

String color;

}

**class** Dog{

**int** age;

String name;

String color;

}

**class** Rabbit{

**int** age;

String name;

String color;

}

这就有一个问题 这三个类，代码重复的部分非常多，怎么解决这个问题呢，就要用到继承了，继承可以帮我们提高代码的复用，当多个类存在相同的属性和方法的时候，可以从这些类抽象出父类，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语句来声明继承了父类就可以拥有这些属性和方法了

class 子类 extends 父类

这样子类就会拥有父类的属性和方法

我们对上面的案例进行改进，我们抽取出相同的属性和方法，封装成一个父类 ，让子类去继承它。

**class** Pet{

**int** age;

String name;

String color;

}

**class** Cat **extends** Pet{

}

子类可以继承父类，那是不是父类所有属性和方法都能被继承呢，我们看下面的案例：

**class** Pet{

**public** **int** age;

**protected** String name;

String color;

**private** **float** price;

}

**class** Cat **extends** Pet{

}

通过该案例，我们了解到，子类继承父类时呢，不能继承父类的private属性和方法，其它修饰符则无影响，可全部继承

继承的特点

1. 子类最多继承一个父类(直接继承)

但 java支持多继承，可通过实现接口（间接继承）来实现多继承

1. 继承关系是传递的，若C继承B，B继承A（此时具有多层继承的关系），则类C具有A和B的属性和方法，
2. java中所有类都是Object的直接或间接子类,基类
3. 子类只能对父类的public、protect和默认的属性或行为进行访问；private属性和方法不能够被继承
4. 子类能够对父类方法进行重写，接下来我们就会讲到

方法的重写和重载

讲解了父类和子类的概念之后呢，我们接下来按照顺序就应该讲多态了，但这里呢，有一个概念要先给同学们讲一下，就是方法的重载和重写，这个面试的时候也经常会被问到

方法的重载：

1. 方法名相同
2. 方法的参数个数，参数类型，顺序至少有一项不同
3. 方法的返回类型可以不同
4. 方法的修饰符可以不同

方法的重写：

方法的重写是子类对父类的方法进行覆盖，叫方法的重写。通过方法的重写，子类可以完成一些父类不具备的功能。

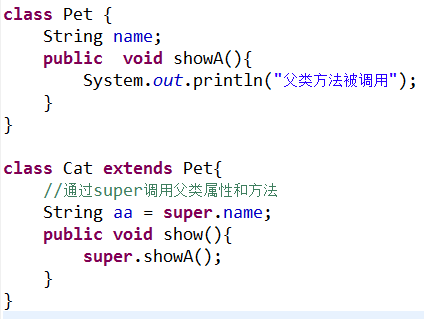
子类重写父类方法时的注意事项：

1. 方法名相同
2. 参数类型，个数，顺序都要保持一致
3. 返回类型要保持一致
4. 子类不可以缩小父类方法的访问权限，但可以扩大
5. 方法体（行为）可以不一样

Super关键字

Super用在子类中代表当前类的父类对象的引用，它有两种使用方法

1. 通过super来该问父类的属性和方法



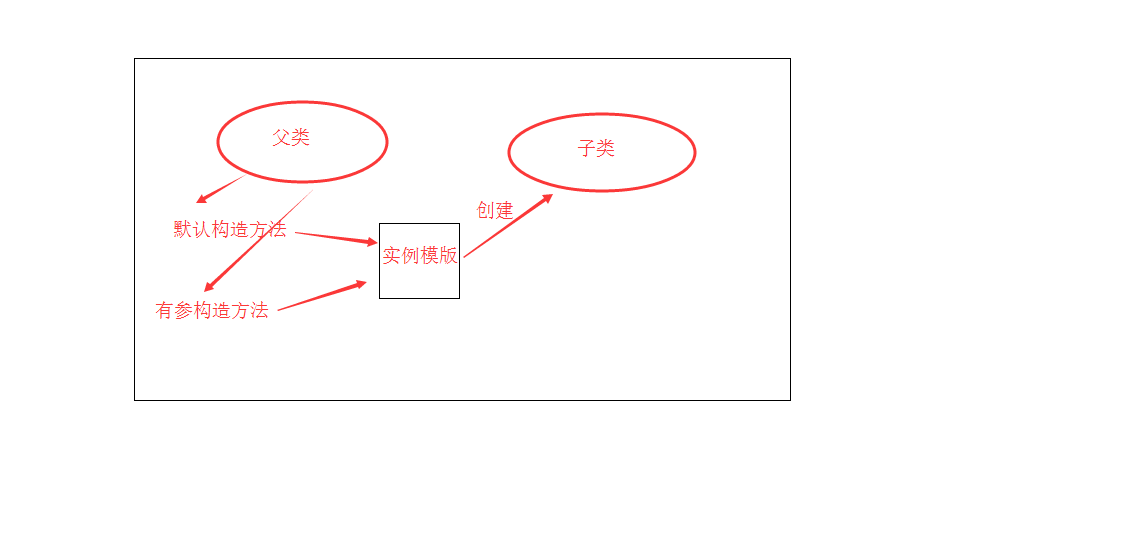
2、通过super来调用父类的构造方法，完成父类的初始化，

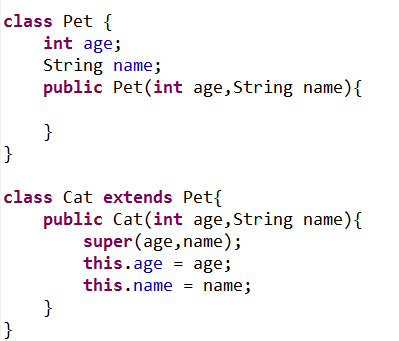
2.1、父类没有定义构造方法，子类可不用显示调用父类构造方法

2.2、父类定义了无参构造方法，子类可不用显示调用父类构造方法

2.3、父类定义了有参构造方法，而没有定义无参构造方法，

这时必须显示调用父类构造方法，对父类进行初始化。 注意: 子类调用父类的构造方法时，super语句必须是子类构造方法的第一句





多 态

Java中有两种情况，可以体现多态，一是继承，二是实现接口。

所谓多态，就是指一种引用(类型)在不同情况下的多种状态，我们也可以这样理解，多态是通过指向父类的指针，来调用不同子类中的实现方法。

我们看下面这个案例

Pet pet1 = **new** Pet();

pet1.cry();

pet1 = **new** Cat();

pet1.cry();

pet1 = **new** Dog();

pet1.cry();

pet1 = **new** Rabbit();

pet1.cry();

注意事项：

父类中的cry()方法虽然没干什么事，但不可以删除，父类可以通过cry()方法来调用子类中被重写的cry()方法，而如果父类中没有这个方法，则无法调用子类中的该方法。

那么多态在我们java编程中有什么作用呢，我们看下面这个案例

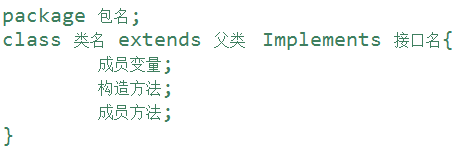
主人给宠物喂食

Eg: Master ms = new Master();

Ms.wei(Pet pet,Food food);

抽象类 接口 final

现在我们已经基本可以构键一个完整的类，



那么下面我们来讲解几种特珠的类，抽象类、接口以及final关键字

一、 抽象类：

为什么需要抽象类，它的作用是什么，

我们思考这么一种情况，我们以前写过一个案例，

**class** Animal{

**public** **int** age;

**public** String name;

**protected** **void** cry(){

System.*out*.println("我是一只动物，我不知道怎么叫唤");

}

}

这个案例中，animal的cry()方法我们都给子类覆盖掉了，这个方法我们基本没用到，而且animal也不知道该怎么叫，像这种父类方法不能确定时，我们可以使用关键字abstract来修饰该方法，这种方法被称为抽象方法，而抽象方法只能放在抽象类中，用abstract修饰的类叫抽象类

我们对上面的代码进行改进

//使用abastract修饰的类

**abstract** **class** Animal{

**public** **int** age;

**public** String name;

//抽象方法，只能放在抽象类中

**public** **abstract** **void** cry();

}

而当我们一个类继承了抽象类之后会发生什么情况呢，我们看以下案例：

//当一个类继承一个抽象类的时候

//java规定，子类需要实现父类中的所有抽象方法

**class** Dog **extends** Animal{

@Override

**public** **void** cry() {

// **TODO** Auto-generated method stub

}

}

抽象注意事项：

1. 用abstract来修饰一个类时，这个类就是抽象类，
2. 用abstract来修饰一个方法时，这个方法就是抽象方法
3. 抽象类不能被实例化(因为方法不完善)
4. 抽象类不一定要包含抽象方法，但抽象方法一定是放在抽象类中
5. 抽象方法不可以有方法体 abstract void cry(){ //不能有大括号 }
6. 抽象类和抽象方法在工作中用的不是很多，但同学们找工作的时候经常会被问到

二、接口

我们先看下面这张图



这就是我们现实生活中的接口，不管是相机，u盘还是手机都可以通过usb接口，连接到我们的电脑。

而我们java中，也给我们提供了接口这么一种机制，让我们可以通过同一个接口，来实现各不同的操作，

接口：

接口就是给出一些没有内容的方法，封装到一起，在某个类要用到的时候再跟据具体情况把这些方法实现.

那我们使用java怎么创建接口，怎么使用接口呢，

**//使用interface关键字来定义一个接口**

**interface** Usb{

**public** **void** connect();

}

**//使用implements关键字来实现接口，**

**//实现一个接口，就要实现该接口中的所有方法**

**class** Photo **implements** Usb{

@Override

**public** **void** connect() {

// **TODO** Auto-generated method stub

System.*out*.println("加载相机的驱动");

}

}

一个类只能继承一个父类，却可以实现多个接口，



接口的注意事项：

1. 接口是一种特殊的抽象类
2. 接口不能被实例化
3. 一个类可以实现多个接口

Class A implements B , C{ }

1. 接口中的方法都不能有方法体，接口中所有方法都是抽象方法，不管加不加abstract关键字
2. 接口中的方法必须是public的，可以使用默认修饰，但编译后会转成public，但不能使用protected和private来修饰
3. 接口中可以有变量，但变量不能用private 和 protected修饰，关于这一点同学们要注意：
   1. 接口中的变量本质上都是static，public的，而且是final类型的，必须初始化，不管你加不加static、final修饰
   2. 在java开发中我们经常把一些常用的变量放在接口中，作为全局变量使用。访问形式：接口名.变量名

6、一个接口不能继承其它类，但可以继承其它接口

7、接口体现了程序设计里的多态。

两个概念：前期绑定和后期绑定

前期绑定：在程序运行之前进行绑定，又叫静态绑定，由编译器和连接程序实现。比如static方法和final方法

Eg: int a = 9;

后期绑定：在运行时跟据对象的类型进行绑定，由方法调用机制实现，叫动态绑定。

Eg: Pet pet = new Cat();

Pet = new Dog();

总结：继承和实现接口的异同：

1、一个类不管是继承父类还是实现接口，都会拥有父类和接口中的属性和方法

2、一个类只能继承一个父类，但可以实现多个接口

3、继承可以向下传递，而接口只对实现该接口的类起作用

4、接口可以看作是对继承的一种补充

Final关键字

final有“这是无法改变的”或者“终级的”含义，它可以修饰非抽象类、非抽象成员方法和变量。

我们在以下情况会用到final

1. 当不希望父类某个方法被子类重写时
2. 当不希望某个变量的值被修改，可以用final
3. 当不希望类被继承时可以在类名前加final

Eg:

//使用final修饰的类不可被继承

**final** **class** Pet{

//final修饰变量(被final修饰的变量就是一个常量)

**final** **float** a = 3.1415926f;

//不希望被子类修改的方法可以用final修饰

**public** **final** **void** showA(){

System.*out*.println(a);

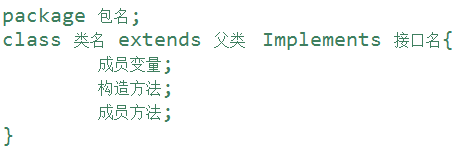
}

}

Final一些特点：

1. final类不能被继承，没有子类，final类中的方法默认是final的。
2. final方法不能被子类的重写，但可以被继承。
3. final成员变量表示常量，只能被赋值一次，定义时必须要初始化，赋值后值不再改变。
4. final不能用于修饰构造方法。

至此我们一个完整的类就给同学们讲解完毕，当然距离我们熟练运用，还有很长的路要走，我们在接下来的学习过程中，慢慢对类和对象，对面象对象编程，就会有一个更加深入的了解，但这个前提就是我们要把面向对象的基础掌握牢固



补充讲解：

1. 全局变量和局部变量

全局变量，是指在类中定义的变量；它在整个类中都有效

全局变量又可分为：类变量（静态变量）和实例变量(差异)

1. 类变量使用static关键词修饰
2. 静态变量是属于类所有，实例变量属于每个实例
3. 类名.属性名(访问静态)

对象名.属性名（访问静态，实例变量）

局部变量：是指那些在方法体中定义的变量以及方法的参数它只在定义它的方法内有效

全局变量在整个类的内部都可以访问，局部变量只在方法内部起作用

思考题：全局变量和局部变量可以同名吗？如果可以，程序会认哪一个