# 第六章 面向对象基础

## 6.1 实现类的基本操作

### 6.1.1 面向对象的基本概念

6.1.1.1· 面向对象

Java是一种面向对象的语言，其基本思想是把问题看成若干个对象组成，这些对象之间是独立的，但有可以互相配合、连接和协调，从而完成整个程序要实现的任务和功能。

面向对象三大特征：封装、继承和多态。

#### 6.1.1.2 对象

用来描述客观事物的一个实体，由一组属性和方法构成。

#### 6.1.1.3 类

具有相同属性和方法的一组对象的集合；

类是对象的抽象，对象是类的具体；

可以认为类是一个模板，通过给模板加入数据，就生成了一个一个对象。

### 6.1.2 定义类

面向对象设计的过程就是定义类的过程，一般份以下三部来进行：

1发现类，类定义了对象的特征和行为（属性和方法）

2发现类的属性，对象所拥有的静态特征在类中表现为属性，如名称、颜色

3发现类的方法，对象执行的操作称为类的方法

#### 6.1.2.1 定义类

定义类的语法格式如下图：



访问修饰符 class 类名（类名首字母大写）

#### 6.1.2.2 属性

属性的语法格式如下图：



访问修饰符 数据类型 变量名

#### 6.1.2.3 方法

方法的语法格式如下图;



访问修饰符 返回类型 方法名 数据类型 参数名

可以不写参数或写多个参数，中间逗号隔开

### 6.1.3 创建和使用对象

#### 6.1.3.1 创建对象

类是一类事物的集合和抽象，代表这类事物共有的属性和行为。

一个对象称为类的一个实例，是类的一次实例化的结果。

类的对象可以调用类中的成员，如属性和方法等。

创建对象的语法格式如下图：



类名 对象名 new 类名

#### 6.1.3.2 使用对象

Java中，要使用对象的行为和属性需要使用”.”。

使用对象的语法格式如下图：



对象名. 方法名



对象名. 属性名

#### 6.1.3.3 对象数组

对象数组是由对象组成的数组，数据类型就是具体的类名。

## 6.2 升级类的功能

### 6.2.1成员方法：

定义在类中的方法。

#### 6.2.1.1带参数的方法

带有参数或由返回值的方法。

方法中由实参和形参：

形参——定义在方法内的参数；

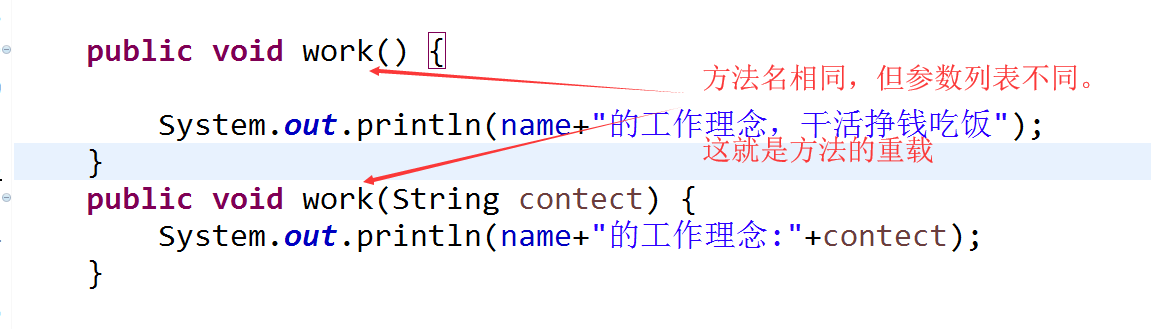
实参——调用方法时传入的参数。

#### 6.2.1.2 方法重载

##### 6.2.1.2.1 方法重载的定义

一个类中定义多个同名的方法，但每个方法具有不同的参数类型或个数。

方法的签名：方法名+参数列表。



##### 6.2.1.2.2 方法重载的特点

·在同一个类中；

·方法名相同；

·参数的个数或类型不同；

·方法的返回值不能作为判断方法之间是否构成重载的依据；

·方法的修饰符(public/private...) 不能作为判断方法之间构成重载的依据；

·构成方法重载后，根据所需要的返回类型和参数列表选择所要使用的方法，方法重载简化了调用方法的代码。

### 6.2.2 成员变量

#### 6.2.2.1 成员变量的作用域

直接定义在类中的变量，定义于方法外部。不赋值时，默认值规则如表6.2：

**表6.2：默认值规则**

|  |  |
| --- | --- |
| 数值类型 | 默认值 |
| byte,short,int,long | 0 |
| flaot,double | 0.0 |
| boolean | false |
| char | /u0000 |
| 引用类型 | null |

#### 6.2.2.2 局部变量的作用域

局部变量就是定义在方法中的变量

#### 6.2.2.3 成员变量和局部变量的区别

1、作用域不同

局部变量只作用于方法内部，该方法外 无法访问；成员变量作用于类中，该类中的所有成员方法都可以调用；如果权限允许，甚至可以作用与类外部。

2、初始值不同

成员变量如果不赋值，会拥有默认的值；局部变量必须先初始化再使用。

3、同一方法中不允许出现同名的局部变量；在不同的方法中可以有同名的局部变量名

4、局部变量名可以和成员变量名同名，且成员变量的优先级更高。

#### 6.2.2.4 数据类型

变量的数据类型分为两种，基本类型和引用类型。

### 6.2.3 构造方法

当类创建一个对象时，会自动调用该类的构造方法；构造方法分为默认构造方法和带参数的构造方法。

#### 6.2.3.1 构造方法的定义

构造方法没有返回值，也不能用void表示；构造方法名与类名一致。其中默认构造方法不带参数。

当没有定义构造方法时，系统会自动添加一个默认构造方法。

定义构造方法的格式如下图：



访问修饰符 方法名（与类名相同） 参数列表

#### 6.2.3.2 构造方法重载

可以写多个构造方法，但参数列表不能相同。

#### 6.2.3.3 this的使用

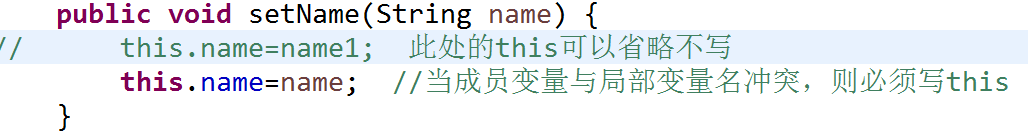
一个对象的默认引用。每个实例方法内部都有一个this引用变量，指向调用 这个方法的对象。

this:指代当前对象

\* 对一个对象的默认引用。

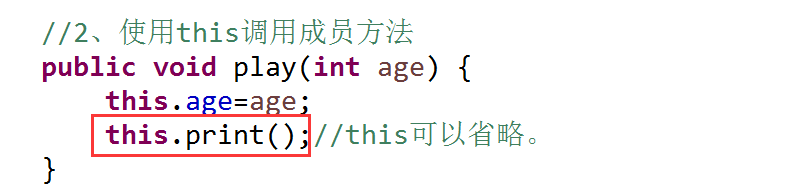
\* 每个实例方法内部都有一个this引用变量，指向调用这个方法的对象

##### 6.2.3.3.1 使用this解决成员变量与局部变量同名冲突的问题

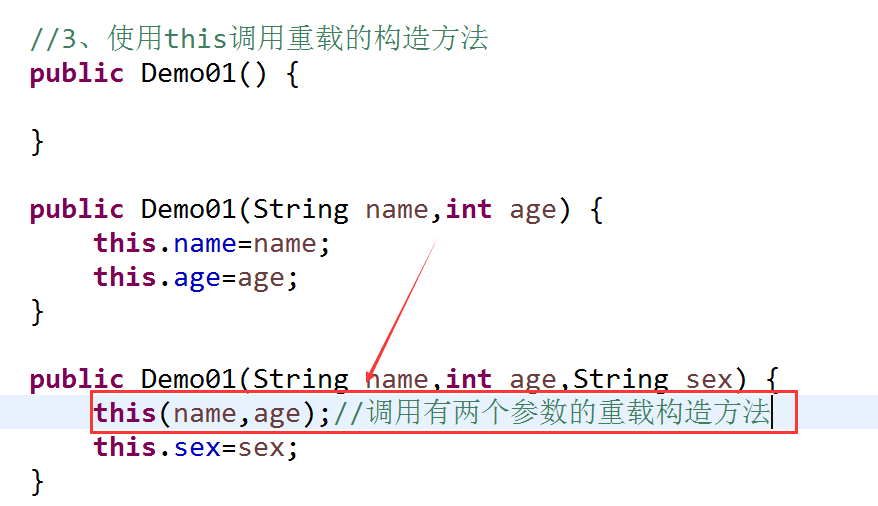


在方法中只要涉及成员变量，都应该加this，只不过当名字不冲突时，可以省略。

##### 6.2.3.3.2 使用this调用成员方法,this可以省略



##### 6.2.3.3.3 this可以调用重载的构造方法



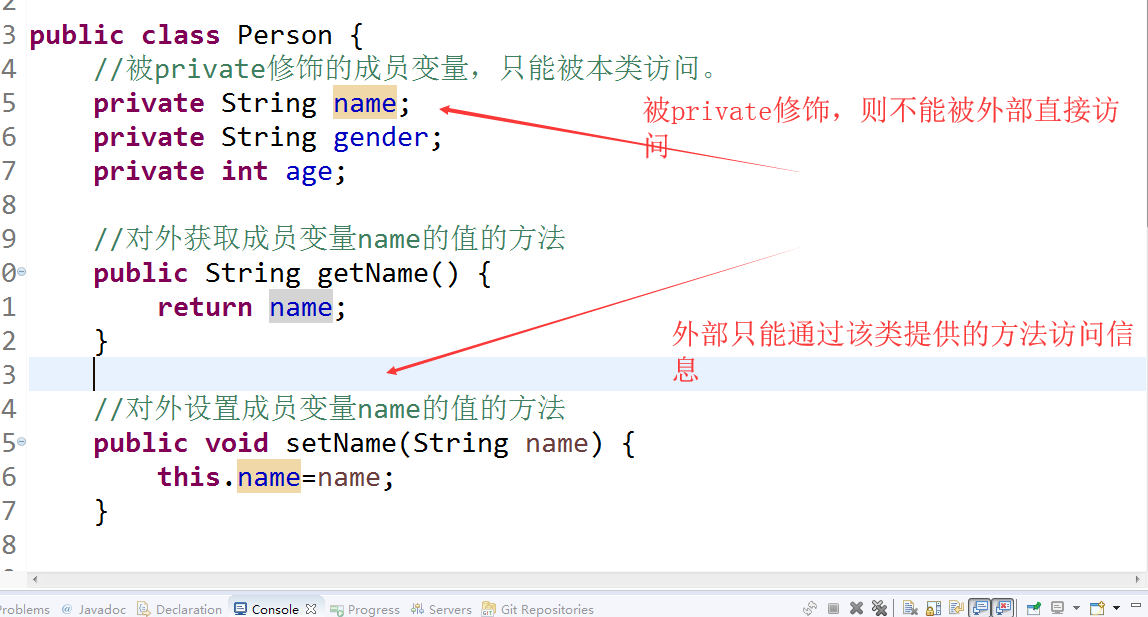
注意：只能在构造方法中使用，且必须放在最前面

## 6.3 封装

### 6.3.1封装概述

其实质是将类的状态，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实现对隐藏信息的操作和访问。

如：



输出结果：



### 6.3.2封装的步骤

1修改属性的可见型

将public改为private

2 设置getter/setter（）方法

手动添加或使用快捷添加。

3设置属性存取限制

利用条件判断语句进行限制

## 6.4package

### 6.4.1 包的定义

package 必须是java源文件的第一条非注释语句。且一个源文件只有一个包声明语句。

编码规范：

1、一个唯一的包名前缀通常是全部小写的ASCII字母(abcd.....)。并且是一个顶级域名com,edu,gov,net,org。通常使用组织的网络域名逆序。

比如：域名为：javagroup.net

包声明： package net.javagroup.mypackage

2、包名后续部分，依据机构内部而决定。

比如： net.javagroup.reseach.powerproject

↑ ↑

项目名 部门名

\* 北大青鸟网址： bdqn.cn 项目名：itrip 部门：tch

\* 包命名： cn.bdqn.itrip.tch 按照规范，这样可以保证包名不重复。

### 6.4.2包的作用：

1、包允许将类组合成较小的单元，易于找到和使用相应的类文件。

2、防止命名冲突：Java中只有在不同的包中的类才能重名。

3、包允许在更广的范围内保护类、数据、方法

### 6.4.3 包的使用——import

实例化Scanner，应该要如下写法：

java.util.Scanner sca=new java.util.Scanner(System.in);

但这么写太麻烦了。

所以可以先导入: **import** java.util.Scanner;

然后就可以直接如下写书：

Scanner input=new Scanner(System.in);

## 6.5 访问修饰符

### 6.5.1 类的访问修饰符

|  |  |  |
| --- | --- | --- |
| 作用域  修饰符 | 同一包中 | 非同一包中 |
| public | 可以使用 | 可以使用 |
| 默认修饰符（不写） | 可以使用 | 不可以使用 |

### 6.5.2 类成员的访问修饰符

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 作用域  修饰符 | 同一类中 | 同一包中 | 子类中 | 其它地方 |
| Private | 可以使用 | 不可以使用 | 不可以使用 | 不可以使用 |
| 默认修饰符（不写） | 可以使用 | 可以使用 | 不可以使用 | 不可以使用 |
| Protected | 可以使用 | 可以使用 | 可以使用 | 不可以使用 |
| Public | 可以使用 | 可以使用 | 可以使用 | 可以使用 |

### 6.5.3 static 关键词

static三种修饰

1、static修饰成员属性：静态变量（类变量），可以直接使用类名调用静态变量

\*被static修饰的变量

\*在内存的静态区/方法区中仅有一个拷贝

\*类内部中，可以在任何方法直接访问静态变量

\*其他类中，也可以直接通过类名访问

2、static修饰方法：称为静态方法/类方法

\*在静态方法中不能直接访问实例变量、实例方法

\*在实例方法中能够调用静态方法 静态变量

3、static块：

\*如果有多个静态块，按顺序加载

\*每个静态代码块只会被加载一次

\*static块属于类的代码块，在类的加载期间执行代码块，只执行一次，可以用来在软件中加载静态资源

\*static块常常用来加载静态资源（图片、音频、视频......）

### 6.5.4 final 关键词

final的三种用法

1、final修饰成员变量和局部变量

1.1、final修饰成员变量

\*意为不可改变

\*两种初始化方法：1、声明的同时初始化；2、构造方法初始化；

1.2、final修饰局部变量

\*意为不可改变

\*在使用之前初始化

2、final修饰方法：不可以被子类重写。

\*其意义为：防止子类定义新方法时，不经意将该方法重写掉

3、final修饰类：意为不可继承。

\*意义在于：保护类不被继承，控制滥用继承对系统造成危害

\*比如String类就是被final修饰

### 6.5.5 static final 修饰的成员变量

\*不能改变，并且唯一

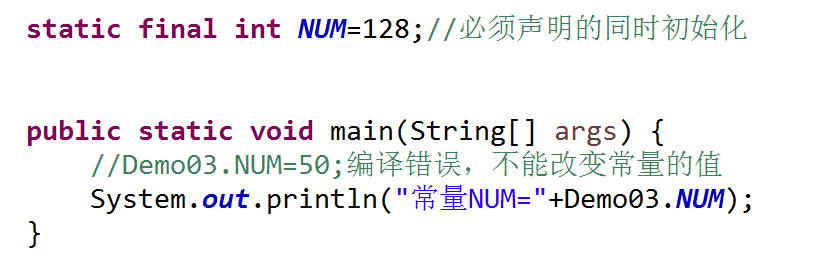
\*必须声明的同时初始化

\*常量的名称建议全部大写

\*在编译期时，被直接替换为具体的值。效率高

\*可以用类名去调用常量

\*在方法区里面，只有一份，不可更改



## 6.7 内存泄漏

内存管理： 由JVM来管理

编译好的java程序需要运行在JVM中。

代码还是数据，都需要存储在内存中，JVM为java程序提供并管理所需要的内存。

JVM内存分为：堆，栈，方法区。分别存储不同的数据

### 6.7.1堆：

1用于存放new出来的对象（包括成员变量）。

2成员变量的生命周期: 通过new创建对象时存在堆中，对象被垃圾回收器GC回收时 一并消失。

3堆中的垃圾： 没有任何引用所指向的对象。

垃圾回收器（GC）不定时的到堆中查看，看到垃圾就自动回收。当然这个回收的过程是透明，并不是一有垃圾就立刻回收。 如果想快点回收垃圾，可以调用System.gc().

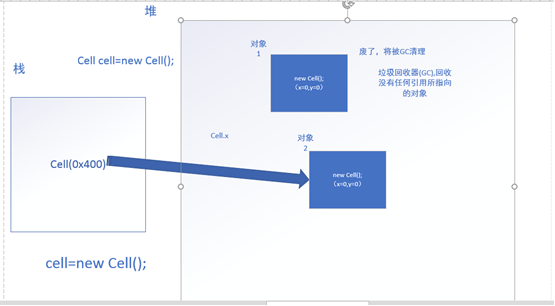
当然这也是建议快点回收垃圾，也不是立马回收。

垃圾回收器：是JVM自带的一个线程。（自动运行的程序），用于回收任何没有引用指向的对象。

因为有垃圾回收器的存在，所以java程序员不用更担心内存管理。

内存泄漏：不在使用的内存，没有被垃圾回收器及时回收。

严重的内存泄漏:会因为过多的内存占用而导致程序崩溃。



当堆中某个对象没有引用，则会被垃圾回收器(GC)，清理

### 6.7.2栈：

1、用于存放正在调用方法中的所有局部变量（包括参数）。

2、调用方法时，在栈中为该方法分配 了一块对应的栈桢，栈桢中包含了所有的局部变量（包含参数）。

方法结束后，栈桢消失，局部变量一并消失。

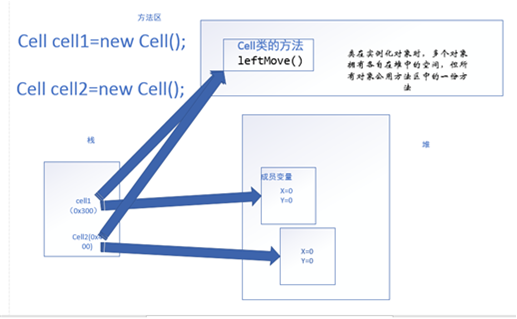
3、局部变量的生命周期： 调用方法时存在栈桢中，方法调用结束时与栈桢一并消失。

### 6.7.3方法区/静态区：

1用于存放.clss字节码文件，其中包含了方法。

java程序运行时，首先通过类装置载入字节码文件，经过解析后将其装入方法区，里面有类的各种信息，包含了方法。

2方法只有一份，通过this来区分具体是哪个引用调用。( 类在实例化对象时，多个对象拥有各自在堆中的空间，但所有对象公用方法区中的一份方法。)



# 第七章 继承和多态

## 7.1继承

7.1.1继承的基本概念

继承是面向对象的三大特征之一，继承可以解决编码中代码冗余的问题，是实现代码重用的重要手段之一。

Java中只支持单继承，即每个类只能有一个直接父类。

继承的语法格式如下图：



访问修饰符 class 子类类名 extends 父类类名

1 可继承内容：

·public、protected修饰的属性和方法，可以跨包继承；

·默认修饰符符修饰的属性、方法，不可跨包；

·无法继承父类的构造方法；

2 特性：

·子类拥有父类非private的属性，方法；

·子类可以拥有自己的属性和方法，可以对父类进行扩展；

·子类可以用自己的方式实现父类的方法；

·Java的继承是单继承，但是可以多重继承 ；

·提高了类之间的耦合性；

·继承使代码可以重用，增加软件的可扩充性。

### 7.1.2 super的三种用法

Super的语法格式如下图：



Super. 方法名/属性

1、访问父类属性

2、访问父类方法

3、访问父类构造方法

注意：

1 使用super关键字,super代表父类对象

2 在子类构造方法中调用且必须是第一句

3 不可以访问父类中定义为private的属性和方法

### 7.1.3 继承条件下构造方法的调用规则

1、子类没有通过super显式调用父类有参构造方法，且没有通过this调用自身构造方法，系统会默认调用父类无参构造方法；

2、如果子类使用super调用了父类有参构造方法，将执行对应的有参构造方法；

3、如果存在多级继承关系，则在实例化一个子类对象时，上述规则将向更高级传递，直到父类为Object；

### 7.1.4 方法重写

子类根据需求对从父类继承的方法进行重新编写；

要求：

1、方法名相同、参数列表相同；

2、返回值类型相同，或者是其子类；

3、重写的方法的访问权限不能缩小；

4、父类是静态方法，重写的方法也必须是静态方法；

5、子类不能抛出比父类更多的异常；

方法重载与方法重写的区别

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 修饰符 | 方法名 | 参数列表 | 返回值类型 | 范围 |
| 重载 | 相同 | 相同 | 不同 | 无关 | 同一个类 |
| 重写 | 不能更严 | 相同 | 相同 | 相同 | 子类 |

## 7.2 多态

### 7.2.1 多态的概念

同一个引用类型，使用不同的实例而执行不同的操作， 呈现出不同的结果；

多态依赖重写实现。

### 7.2.2 向上转型

子类向父类转型称为向上转型。

格式：

父类型 引用变量名=new 子类型()；

规则：

1、向上转型时，子类型自动转为父类型；

2、通过父类引用变量调用子类重写的方法，而不是父类的方法；

3、子类特有的方法，不能通过父类引用变量调用。

### 7.2.3 向下转型

将父类对象赋值给子类，即父类转换为子类（需要强制转换）。

格式：

子类型 引用变量=(子类型) (父类型引用变量)；

instanceof：

解决了在向下转型的过程中，如果不是转换为真实的子类，则编译错误，可以使用instanceof关键词，实现预判下是否能转换为其子类。

语法：父类引用变量 instanceof 子类型