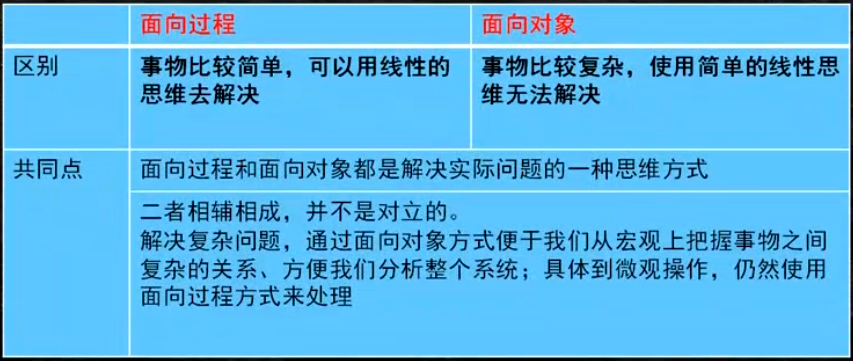
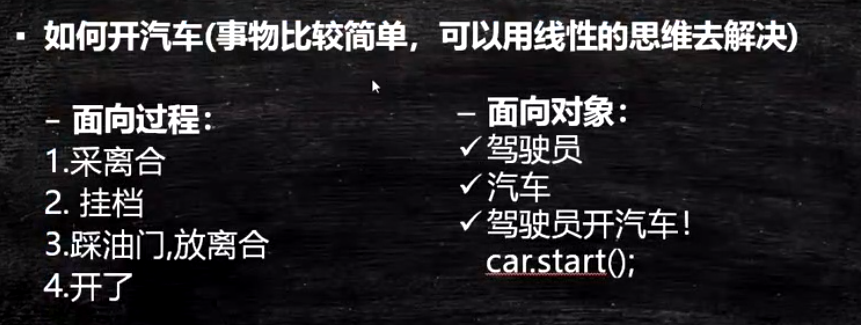
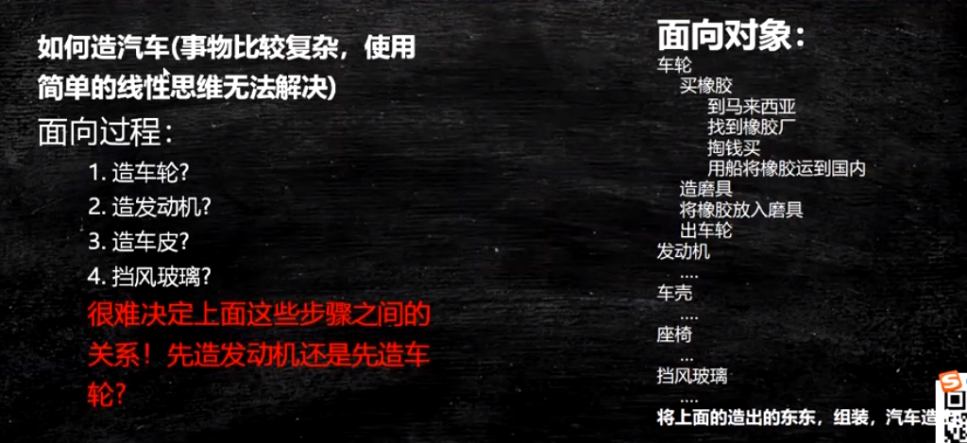
面向对想想编程

（OOP：Object Oriented Programming）





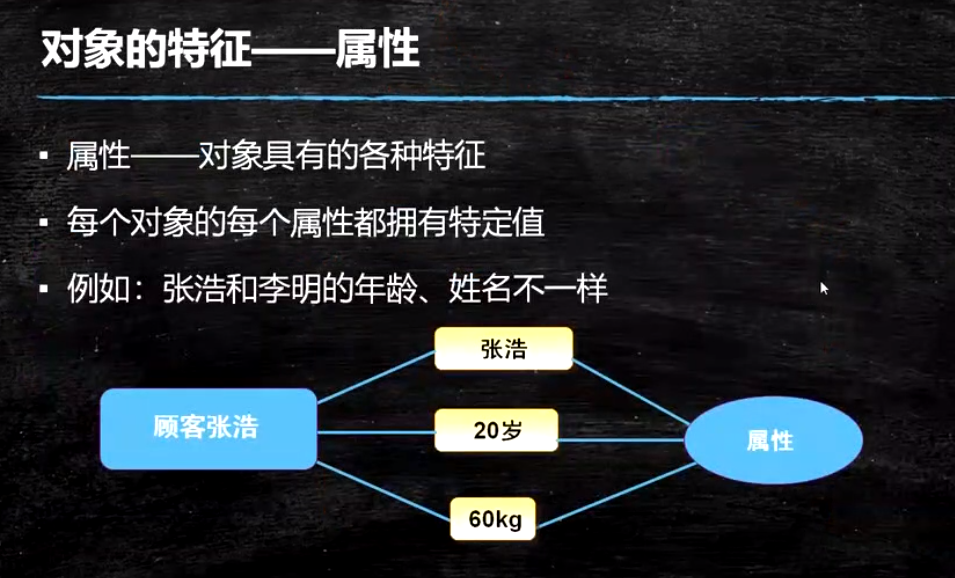


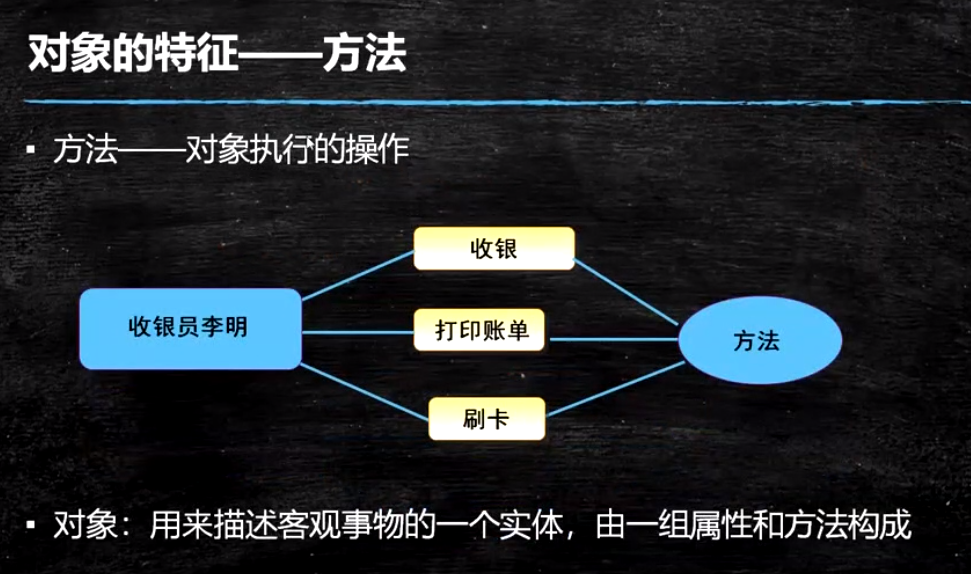
* 类可以生成对象，类是对象的抽象

先有具体的对象，然后抽象各个对象之间像的部分，归纳出类，通过类再认识其他对象

类是具有相同属性和方法的一组对象的集合（抽象）。类是对象的类型，不同于int类型：具有方法

比如只说“杯子”不是一个对象，是对象的抽象（即类）。因为只说杯子不知道是玻璃杯，保温杯还是什么具体的杯，这个具体的杯被称为是对象。现实世界由**对象**组成，因为真正看得见摸得着的是实实在在的对象





方法和函数表示的意义本质上是不一样的，它们实现的功能是一样的，但是Java里没有函数（function），只有方法（method）

-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

属性：即每个对象都有ta自己的特定的值为**属性**（如人有姓名，身高，体重等），而ta

的行为动作为**方法**（如人需要吃喝拉撒睡等）

⭐写程序的时候如果一个逻辑代码块写到类里面它就是个方法，如果写到类外面它就是个函数。如在Python里进去文件就可以直接写代码，不需要定义Public class类，这写的就是函数。而在Java里进去要先定义类，然后在类的里面写，这时写的就是方法





* 总结

-对象和类的关系：特殊到一般，具体到抽象

-类：我们叫做class

-对象：我们叫做Object，instance（实例）。以后我们说某个类的对象，某个类的实例，是一样的意思

-类可以看成一类对象的模板，对象可以看成该类的一个具体实例

-类是用于描述同一类型的对象的一个抽象的概念，类中定义了这一类对象所具有的静态和动态的属性

-对象是Java程序的核心，在Java程序中“万事万物皆对象”

-JDK提供了很多类供编程人员使用，编程人员也可以定义自己的类

-所有类定义的时候可以添加属性和方法，但不是必须要写的

**类的属性**

属性field，或者叫成员变量

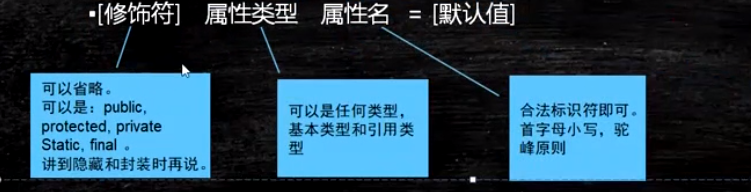
属性用于定义该类或该类对象包含的数据或者说静态属性

属性作用范围是整个类体

在定义成员变量时可以对其初始化，如果不对其初始化，Java使用默认的值对其初始化（

数值：0，0.0 char：\u0000 boolean: false 所有引用类型：null）

属性定义格式：



IDE(Integrated Development Environment),集成开发环境

**/\***

**属性：  
语法：  
[访问修饰符] 数据类型 属性名称=值;  
注意：  
 定义属性的时候可以有值也可以没有值  
 但必须要包含类型和名称**

**所有Java程序都以类class为组织单元**

**关键字class定义自定义的数据类型**

**一个java文件中可以定义N多个class，但是只能有一个public class，并且public class的类名要和文件名保持一致**

**\*/  
  
public class** Student {  
 **//属性的定义  
 int stuNumber**;  
 String **name**;  
 **int age** = 20;  
}

面向对象中，整个程序的基本单位是类，方法是从属于对象的

**/\***

**方法：表示行为  
语法：  
 [访问修饰符] 返回值类型（任何类型） 方法名称（形参列表）{   
 逻辑代码**

**}  
假设炒一盘菜，形参列表相当于食材，逻辑代码相当于炒的过程，结果相当于这盘菜  
  
注意：  
 方法的访问修饰符可以不写  
 方法可以有返回值，也可以没有返回值，void表示没有返回值的意思  
 形参列表可以有，也可以没有**

**JAVA中方法参数传递是：值传递**

**\*/**

❀不带形参列表的定义方法

**void** study(){  
 System.***out***.println(**"我正在学习"**);  
}  
❀带形参列表的定义方法

**void** eat(String food){  
 System.***out***.println(**"我正在吃"**+food);  
}

⭐这时这两个方法都无法运行，方法想要运行必须要有**main函数**

类将现实世界中的概念模拟到计算机程序中

**public class 类名{**

//**定义属性部分**

属性1的类型 属性1;

属性2的类型 属性2;

…

属性n的类型 属性n;

//**定义方法部分**

方法1;

方法2;

…

方法m：

**}**

**class** Person{  
 **//成员变量** String **name**; **//姓名  
 int age**; **//年龄** String **sex**; **//性别  
  
 //定义类的introdu()方法，用于输出类相关信息  
 public void** introduce() {  
 System.***out***.println(**"name="**+**name**+**"\tage="**+ **age**+**"\tsex="**+**sex**);  
 }  
}

**/\***

**对象的创建和使用  
 使用new关键字来创建对象  
 类名 对象名 = new 类名();**

**Person person = new Person();  
 对象的使用  
 使用对象的时候可以通过 对象名称.属性 或者 对象名称.方法 来使用属性和方法  
   
  
 \*/  
  
public class** Student {  
 **//属性的定义  
 int stuNumber**;  
 String **name**;  
 **int age** = 20;  
  
 **void** study(){  
 System.***out***.println(**"我正在学习"**);  
 }  
  
 **void** eat(String food){  
 System.***out***.println(**"我正在吃"**+food);  
 }  
  
 **public static void** main(String[] args) {  
 **//创建对象** Student student = **new** Student();  
 **//使用属性** System.***out***.println(student.**name**);  
 System.***out***.println(student.**age**);  
 System.***out***.println(student.**stuNumber**);

**/\*修改属性值，给属性赋值：**

student.**name**=**"张三"**;  
student.**age**=40;  
student.**stuNumber**=20190801;  
System.***out***.println(student.**name**);  
System.***out***.println(student.**age**);  
System.***out***.println(student.**stuNumber**);

**\*/**  **//调用方法** student.study();  
 student.eat(**"apple"**);  
 }  
}

⭐关于String字符比较的问题：

== 比较的是地址

equals 比较的是具体的值

**import** java.util.Scanner;  
  
**public class** StringDemo {  
 **public static void** main(String[] args) {  
 String str = **"abc"**;  
 String str1 = str;  
 System.***out***.println(str == str1);  
 System.***out***.println(str.equals(str1));  
  
 Scanner input = **new** Scanner(System.***in***);  
 String str2 = input.nextLine();  
 System.***out***.println(str == str2);  
 System.***out***.println(str.equals(str2));  
  
 String str3 = **new** String(**"abc"**);  
 System.***out***.println(str == str3);  
 System.***out***.println(str.equals(str3));  
 }  
}

这里结果为

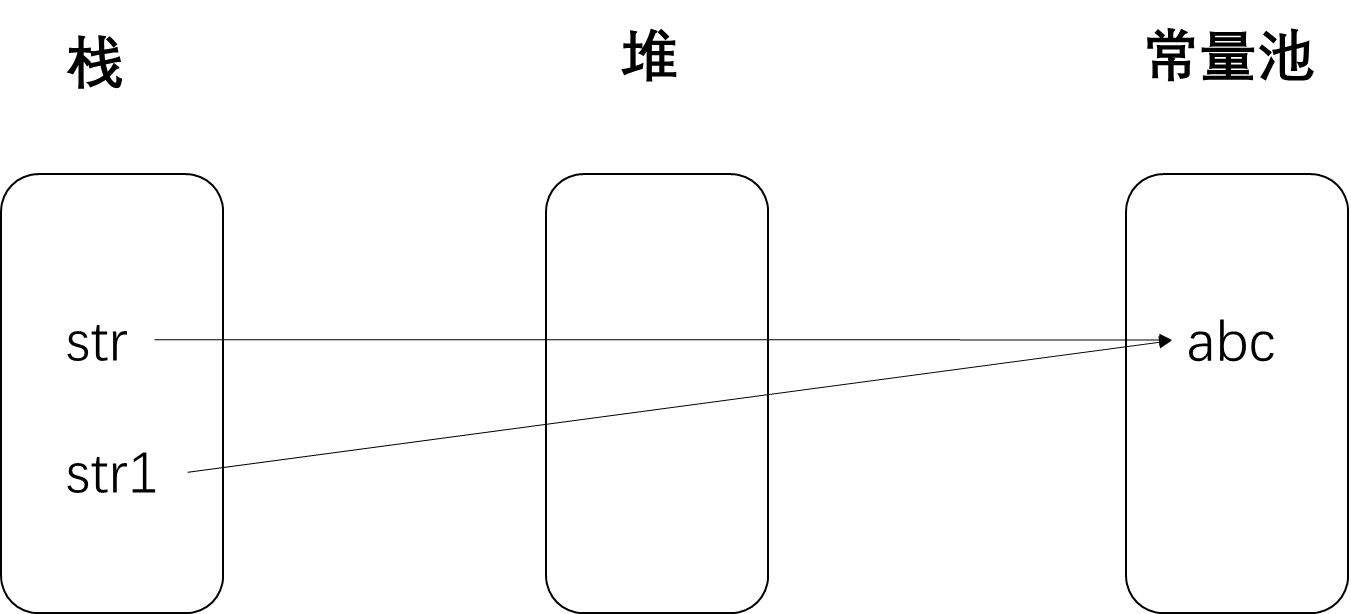
第一组：true，true

第二组：输入abc之后，false，true

第三组：false，true

原因：

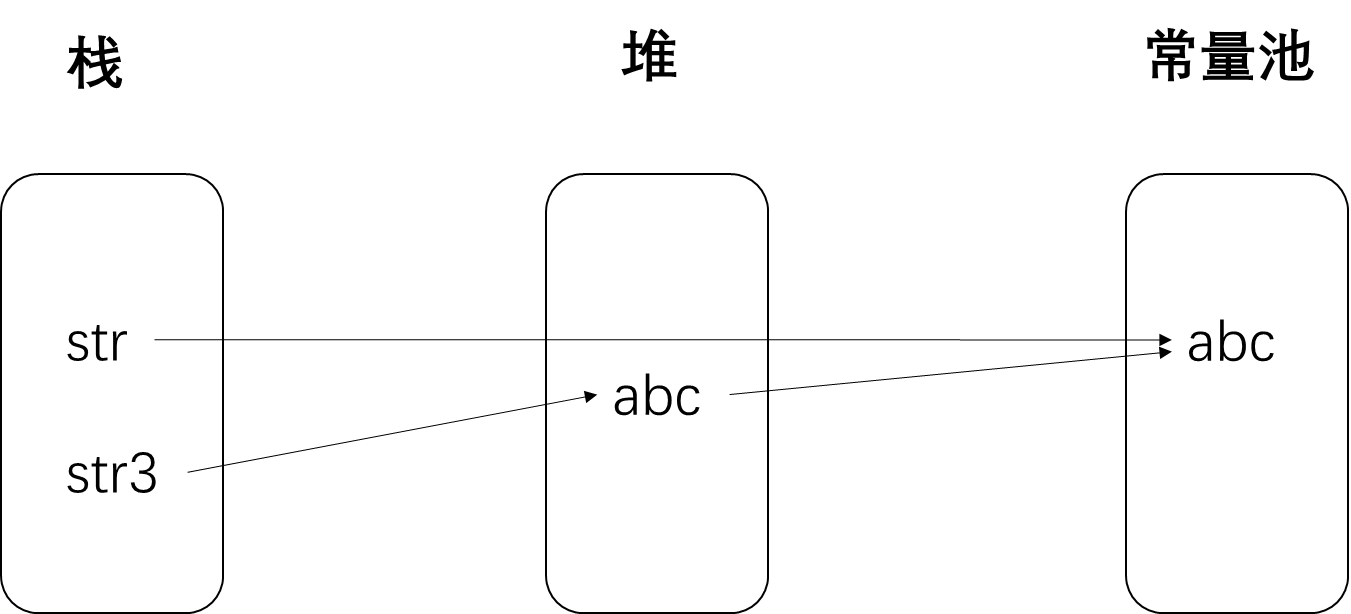
字符串在内存中存放的地方叫常量池。字符串的值是不可以修改的



第一组：

定义str为abc指的是在常量池里的地址，而str1等于str，即str1也指向这个地址。所以它俩无论是地址还是值都是相等的，因此都为true

第二组与第三组：



只要是new的对象，它都会进入到堆里。也就是刚开始str的地址指向的是常量池里的地址，而str3和str2指向的是堆里的地址，==相当于两个不同的地址进行比较，因此为false

只要是用new()来新建对象的，都会在堆中创建，而且其字符串是单独存值的，即使与栈中的数据相同，也不会与栈中的数据共享

* 定义管理员类

1. 定义管理员类Administrator
2. 定义其属性和方法
3. 定义测试类TestAdministrator
4. 创建两个管理员类的对象，并输出他们的信息
5. 输入旧的用户名和密码，如果正确，方有权限更新
6. 从键盘获取新的密码，进行更新

**import** java.util.Scanner;  
  
**public class** Administrator {  
 String **loginName** = **"admin"**;  
 String **password** = **"123"**;  
  
 **public void** introduce(){  
 System.***out***.println(**loginName**);  
 System.***out***.println(**password**);  
 }  
  
 **public static void** main(String[] args) {  
 Administrator admin = **new** Administrator();  
 System.***out***.println(admin.**loginName**);  
 System.***out***.println(admin.**password**);  
 admin.introduce();  
 **//从控制台读取数据，需要Scanner对象** Scanner input = **new** Scanner(System.***in***);  
 System.***out***.println(**"修改用户名及密码"**);  
  
 **while**(**true**){  
 System.***out***.println(**"请输入用户名"**);  
 String name = input.nextLine();  
 System.***out***.println(**"请输入密码"**);  
 String pwd = input.nextLine();  
  
 **if** (name.equals(admin.**loginName**) && pwd.equals(admin.**password**))

{  
 System.***out***.println(**"请输入新用户名"**);  
 admin.**loginName** = input.nextLine();  
 System.***out***.println(**"请输入新密码"**);  
 admin.**password** = input.nextLine();  
 System.***out***.println(**"修改成功"**);  
 **break**;  
 }**else**{  
 System.***out***.println(**"用户名或密码不正确，请重新输入"**);  
 }  
 }  
  
 admin.introduce();  
 }  
 }

**成员变量和局部变量**

变量是用来存储数据的，变量只能存储单一的值，对象能存不同类型的多个值

* 区别

-声明位置不同： 类中；方法中

-作用范围不同：当前类的方法；当前方法

不同的方法中即使有同名的局部变量，没有关系，互不影响，建议相同

-内存存放的位置：堆内存中；栈内存中

-成员变量有默认值；局部变量没有默认值

**/\*  
变量  
 局部变量：  
 定义在方法中的变量称之为局部变量  
 作用域：从定义的位置开始到整个方法结束  
 注意：局部变量只能在当前方法中使用，其他地方无法使用  
  
 成员变量：  
 定义在方法外，类内的变量，也可以叫做全局变量  
 作用域：整个类体内  
 \*/  
  
public class** VarDemo {  
 **static int** *age* = 20;  
 **public void** show(){  
 **int** a = 10;  
 String name = **"zhangsan"**;  
 }  
  
 **public static void** main(String[] args) {  
 System.***out***.println(a);  
 System.***out***.println(*age*);  
 }  
}

这里的**static int** *age* = 20;是成员变量，

**int** a = 10;  
String name = **"zhangsan"**; 是局部变量

**public class** VarDemo {

**static int** *age* = 20;  
  
 **public void** test(){  
 System.***out***.println(*age*);  
 *age* = 10;  
 System.***out***.println(*age*);  
 **int** age = 30;  
 System.***out***.println(age);  
 }  
 **public static void** main(String[] args) {  
 System.***out***.println(*age*);  
 **new** VarDemo().test();  
 System.***out***.println(*age*);  
 }  
}

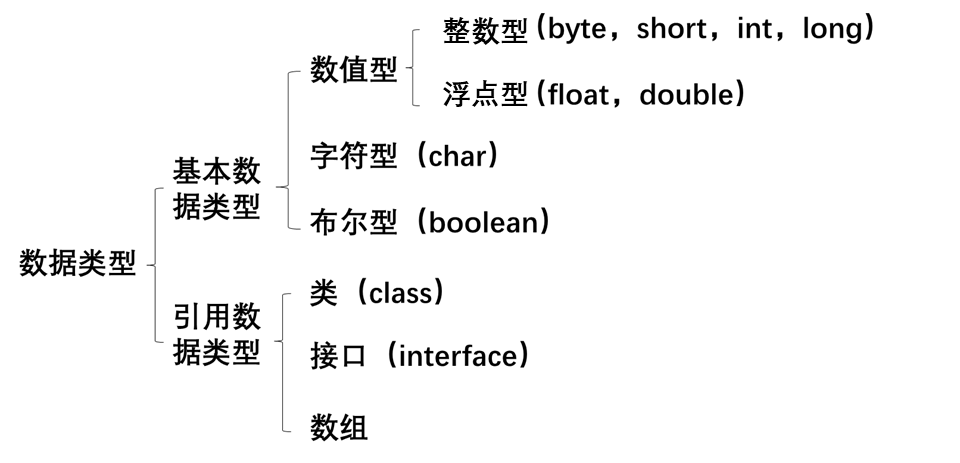
执行结果为：20，20，10，30，**10**

在成员变量定义的age=20和在方法里定义的age=30这两个age并不是同一个变量

age=30只在自己的方法里有效。而刚开始等于20，在执行了test（）方法以后先将age的值从20改为了10，因此最后打印的效果为10

**引用类型**

Java语言中除基本类型之外的变量类型都称之为引用类型



Java中的对象和数组是通过引用对其操作的

-引用可以理解为一种受限的指针

-指针是可以进去与整数做加减运算的，两个指针之间也可以进行大小比较运算和相减运算。引用不行，只能进行赋值运算。因为它里面只能保存地址，通过这个地址找到内存空间中的值

-引用就是一个变量或对象的别名（引用的本质是一个对象）；指针是一个段内存空间的地址（指向存储一个变量值的空间或一个对象的空间）

比如我现在定义了一个 int i=10 ， 这时这个i是在栈里的

我现在又定义了一个int[] arr=new int[5] , 这时这个变量arr是在栈里的，但是new出来的实际对象是在堆里

目前可以理解为基础类型都是在栈里的，而实际内容在堆里的为引用类型

**内存分析**

* 栈(stack)：

存放局部变量

先进后出（FILO first in last out），自下而上存储

方法执行完毕，自动释放空间

* 堆(heap)：

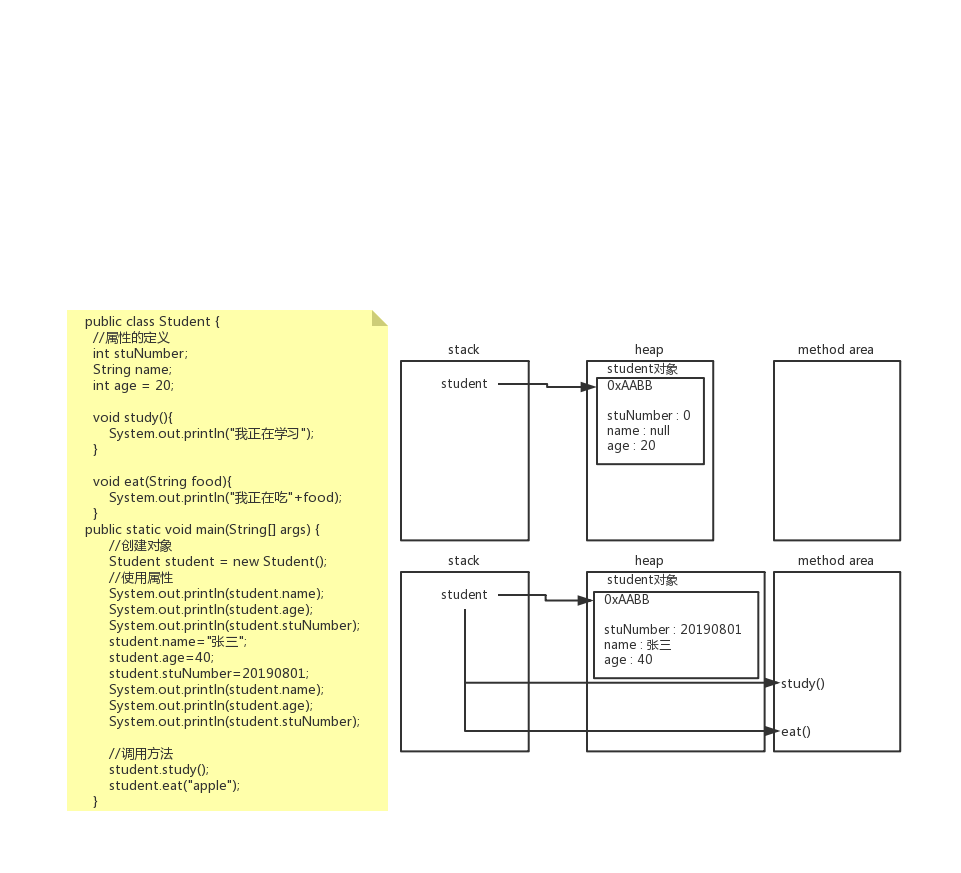
存放new出来的对象

需要垃圾回收器来回收：System.gc()

如果栈里面没有一个指针指向堆里的一个对象了，那么这个对象就可以被回收了

* 方法区：

存放：类的信息(代码)，static变量，字符串常量等



**构造方法**

* 概念：

创建对象的时候默认会调用构造方法来创建对象(在堆中开辟空间)，可以完成成员变量的某些初始化操作

* 语法：

方法名称：构造方法的方法名称必须跟类的名称保持一致

访问修饰符：

形参：可以用户自定义添加，跟方法的普通参数一样

方法体：完成对象的初始化功能

返回值：没有返回值

* 注意：

-创建完类之后，如果没有手动调用构造方法，会有一个默认的无参的构造方法供调用

-当用户自定义了构造方法之后，默认的无参构造方法就不能够使用了，必须要手动定义无参构造方法

-同一个类中可以包含多个同名的构造方法

**public class** Teacher {  
  
 String **name**;  
 **int age**;

**//构造方法（不带参数的）**  
 **public** Teacher(){  
 }  
  
  
 **//构造方法（带参数的）**

**/\*直接在方法的参数里给属性赋值，本例中xname(局部变量)的值赋给name(成员变量)，xage(局部变量)的值赋给age(成员变量)，在main方法里new出对象的时候，直接在括号中把参数写上就直接可以给属性赋值了**

**\*/**

**注意：构造方法一定不能出现void**

**public** Teacher(String xname,**int** xage){  
 System.***out***.println(**"new..."**);  
 **name** = xname;  
 **age** = xage;  
 }  
  
 **public static void** main(String[] args) {  
 Teacher teacher1 = **new** Teacher();  
 teacher1.**name** = **"Lucy"**;  
 teacher1.**age** = 18;  
 System.***out***.println(teacher1.**name**);  
 System.***out***.println(teacher1.**age**);  
 Teacher teacher2 = **new** Teacher(**"Lily"**,29);  
 Teacher teacher3 = **new** Teacher(**"Mike"**,33);  
 System.***out***.println(teacher2.**name**);  
 System.***out***.println(teacher2.**age**);  
 System.***out***.println(teacher3.**name**);  
 System.***out***.println(teacher3.**age**);  
 }  
}

执行结果：

Lucy

18

new...

new...

Lily

29

Mike

33

**重载**

在一个类中可以包含多个重名的方法，但是方法的参数列表不能相同

三个方面的不同：

-参数的个数不同

-参数的类型不同

-参数的顺序不同

注意：

一般构造方法都会进行重载（一个类中可能包含多个属性值，当只需要给部分属性初始化的时候需要调用不同的构造方法）

**public class** Teacher {  
  
 String **name**;  
 **int love**;  
 String **sex**;  
  
 **public void** print(){  
 System.***out***.println(**"name="**+**name**+**"\tlove="**+**love**+**"\tsex="**+**sex**);  
 }  
  
 **public** Teacher() {  
 **name** = **"qq"**;  
 **love** = 20;  
 **sex** = **"Q仔"**;  
  
 }  
  
 **public** Teacher(String name, **int** love, String sex) {  
 **this**.**name** = name;  
 **this**.**love** = love;  
 **this**.**sex** = sex;  
 }  
  
 **public static void** main(String[] args) {  
 Teacher teacher1 = **new** Teacher();  
 teacher1.print();  
 Teacher teacher2 = **new** Teacher(**"美美"**,20,**"Q仔"**);  
 teacher2.print();  
 }  
  
}

运行结果：

name=qq love=20 sex=Q仔

name=美美 love=20 sex=Q仔

**构造器**

定义：

Constructor构造方法，一个在创建关键对象时被自动调用的特殊方法

作用：

为对象进行初始化（成员变量）工作

注意：

-构造器的方法名必须和类名一致

-构造器虽然有返回值，但是不能定义返回类型（返回值的类型肯定时本类），**不能**在构造器里调用**return**

-通过new关键字调用

-如果我们没有定义构造器，则系统会自动定义一个无参的构造方法。如果已定义则编译器不会添加无参数构造方法

-与普通方法一样，构造方法也可以重载

* 定义一个”点”(Point)类用来表示二维空间中的点（有两个坐标），要求如下：

-可以生成具有特定坐标的点对象

-提供可以设置两个坐标的方法

-提供可以计算该点距另外点距离的方法

**public class** Point {  
  
 **int x**;  
 **int y**;  
  
 **//设置二维的坐标点  
 public void** set(**int** xx,**int** yy){  
 **x** = xx;  
 **y** = yy;  
 }  
  
 **public double** calcDistance(Point p){  
 **return** Math.*sqrt*((p.**x**-**x**)\*(p.**x**-**x**)+(p.**y**-**y**)\*(p.**y**-**y**));  
 }  
  
 **public static void** main(String[] args) {  
 Point p1 = **new** Point();  
 p1.set(3,5);  
 Point p2 = **new** Point();  
 p2.set(2,4);  
 System.***out***.println(p1.calcDistance(p2));  
 }  
  
}

执行结果：1.4142135623730951

* 实现交换两个坐标的值：
* **public class** Point {  
    
   **int x**;  
   **int y**;  
    
   **//设置二维的坐标点  
   public void** set(**int** xx,**int** yy){  
   **x** = xx;  
   **y** = yy;  
   }  
    
   **public void** changePoint(Point p){  
   **int** a = p.**x**;  
   p.**x** = **x**;  
   **x** = a;  
   **int** b = p.**y**;  
   p.**y** = **y**;  
   **y** = b;  
   }  
    
   **public void** show(){  
   System.***out***.println(**"["**+**x**+**","**+**y**+**"]"**);  
   }  
    
   **public static void** main(String[] args) {  
   Point p1 = **new** Point();  
   p1.set(3,5);  
   Point p2 = **new** Point();  
   p2.set(2,4);  
   p1.changePoint(p2);  
   p1.show();  
   p2.show();  
   }  
    
  }

执行结果：

[2,4]

[3,5]

* 比较两个人的年龄差：

**public class** Person {  
  
 **int age**;  
 String **name**;  
  
 **public** Person(**int** age){  
 **this**.**age** = age;  
 }  
  
  
 **public int** diffage(Person person){  
 **return** person.**age**-**this**.**age**;  
  
 }  
  
 **public static void** main(String[] args) {  
 Person person1 = **new** Person(20);  
 Person person2 = **new** Person(41);  
 System.***out***.println(person1.diffage(person2));  
 }  
  
}

执行结果：21

this关键字

作用：

this表示的是当前对象本身，更确切地说，this代表当前对象的一个引用

普通方法中使用this：

当多个普通方法之间需要调用的时候，可使用this来进行调用，指的是当前对象的其他方法（这时this可以省略）

-区分类成员属性和方法的形参

-位置：任意

构造方法中使用this：

当构造方法中的参数名称和类的成员变量名称一样的时候可以使用this代表当前对象

当构造方法中需要调用其他构造方法时，可以使用this()，相当于this.ThisDemo()，但必须位于方法体的第一行

注意:有了this之后，可以将构造方法的参数和成员变量保持一致

-使用this来调用其他构造方法

-位置：必须是第一条语句

this不能用于static方法

成员变量的使用：

当方法中的参数名称和成员变量保持一致的时候，使用(this.变量名称)表示的是对象的值，而使用变量名称表示形参列表的值

**public class** ThisDemo {  
  
 String **name**;  
 **int age**;  
  
 **public** ThisDemo(String name,**int** age){  
 **this**.**name** = name;  
 **this**.**age** = age;  
 }  
  
 **public void** test1(){  
 System.***out***.println(**"test1"**);  
 }  
  
 **public void** test2(String name){  
 System.***out***.println(**"test2"**);  
 **this**.test1();  **//这个this.可有可无**  
 System.***out***.println(name);  
 System.***out***.println(**this**.**name**);  
 }  
  
 **public static void** main(String[] args) {  
 ThisDemo td = **new** ThisDemo(**"zhangsan"**,12);  
 System.***out***.println(td.**name**);  
 System.***out***.println(td.**age**);  
 td.test2(**"lisi"**);  
 }  
}

执行结果：

zhangsan

12

test2

test1

lisi

zhangsan

⭐只要是在当前类里面用的（this.变量名）,this都是表示同一个，并且有且只有一个

**public** ThisDemo(String name){  
 **this**.**name** = name;  
}  
  
**public** ThisDemo(String name,**int** age){  
 **this**.**name** = name; **// 也可以写成 → this(name);  
 this**.**age** = age;  
}

**static关键字**

在类中，用static声明的成员变量为静态成员变量，或者叫做：类属性，类变量

它为该类的公用变量，属于类，被该类的所有实例共享，在类被载入时被显示初始化，也就是其他的成员变量要先创建对象才能用，但是static成员变量不需要创建对象就可以使用

对于该类的所有对象来说，static成员变量只有一份，被该类的所有对象共享

普通变量在使用的时候，必须要通过对象名进行调用，类变量可以使用“对象.类属性”来调用，也可以使用”类名.类属性”， 但是推荐使用”类名.类属性”来调用

成员变量放在堆中，static变量置于方法区中

**public class** StaticDemo {  
  
 **/\*定义的name是成员变量，只能通过（对象.类属性）调用，这里相当于**

**staticDemo.name，因此StaticDemo.name部分的name变红了  
 定义的age是静态成员变量，可以通过（对象.类属性），也可以通过（类名.类属性）调**

**用，这里相当于staticDemo.age和StaticDemo.age  
 \*/** String **name** = **"zhangsan"**;  
 **static int** *age* = 10;  
  
  
 **public static void** main(String[] args) {  
 StaticDemo staticDemo = **new** StaticDemo();  
 **//使用对象进行调用** System.***out***.println(staticDemo.**name**);  
 System.***out***.println(staticDemo.*age*);  
 **//使用类名进行调用** System.***out***.println(StaticDemo.**name**);  
 System.***out***.println(StaticDemo.*age*);  
 }  
  
}

用static声明的方法为静态方法或者类方法

静态变量不能定义在静态方法中

不需要对象，就可以调用(类名.方法名)

在调用该方法时不会将对象的引用传递给它，所以在static方法中不可访问非static的成员

静态方法不能以任何形式引用this和super关键字，因为static在创建对象之前已经有了，因此在调this的时候没有这个对象，无法引用

一般工具类中的方法定义为static

**public class** StaticDemo {  
   
 **static int** *age* = 10;  
  
 **public static void** main(String[] args) {  
 StaticDemo staticDemo = **new** StaticDemo();  
 System.***out***.println(staticDemo.*age*);  
  
 staticDemo.*age* = 20;  
 System.***out***.println(staticDemo.*age*);  
 System.***out***.println(StaticDemo.*age*);  
  
 StaticDemo.*age* = 30;  
 System.***out***.println(staticDemo.*age*);  
 System.***out***.println(StaticDemo.*age*);  
  
 StaticDemo staticDemo1 = **new** StaticDemo();  
 System.***out***.println(staticDemo1.*age*);  
   
 }  
  
}

执行结果：

10

20

20

30

30

30

**public class** StaticDemo {  
  
 **static int** *age* = 10;  
  
 **public static void** test(){  
 System.***out***.println(**"static"**);  
 }  
  
 **public void** test2(){  
 *test*(); **//**⭐**静态方法可以在非静态方法中进行调用**  
 }  
  
 **public static void** main(String[] args) {  
 StaticDemo sd = **new** StaticDemo();  
 sd.*test*();  
 StaticDemo.*test*();  
 sd.test2();

}

}

执行结果：

static

static

static

**public static void** test(){  
 test2(); **//这样在静态方法里调用动态方法不行**  
}  
  
**public void** test2(){  
 System.***out***.println(**"static"**);  
}

**public static void** test(){  
 **new** StaticDemo().test2(); **//这样就可以了**  
}  
  
**public void** test2(){  
 System.***out***.println(**"static"**);  
}

⭐静态方法中不能直接调用非静态方法，想使用必须要new一下

static修饰与非static修饰的区别：



**代码块**

概念：使用”{}”括起来的一段代码

分类：根据位置可分类

* 普通代码块→直接在方法或语句中定义的代码块

**public class** CodeBlockDemo {  
  
 **public void** test(){  
 System.***out***.println(**"test"**);  
 {  
 System.***out***.println(**"我应该是什么分类"**);  
 }  
 }  
  
 **public static void** main(String[] args) {  
 **new** CodeBlockDemo().test();  
 {  
 System.***out***.println(**"main"**);  
 }  
 }  
  
}

执行结果：

test

我应该是什么分类

main

* 构造代码块→直接写在类中的用{}括起来的代码块

注意：每次代码运行的时候会将构造代码块中的代码添加到构造方法的前面

构造代码块中的代码会添加到每一个构造方法中，当使用this(参数)的时候不会

添加

**public class** CodeBlockDemo {

**public** CodeBlockDemo(){  
 System.***out***.println(**"构造方法"**);  
}

**//↓的这个构造代码块用反编译软件编译以后发现它给自动加上了**

**public CodeBlockDemo**

**即最后的效果是↓**

**public class CodeBlockDemo{**

**public CodeBlockDemo(){**

**System.out.println("构造代码块");**

**System.out.println("构造方法");**

**}**

{  
 System.***out***.println(**"构造代码块"**); **//这个花括号的整个部分不论写在**

**CodeBlockDemo的什么位置都会第**

**一个打印出来** }

**public void** test(){  
 System.***out***.println(**"test"**);  
 {  
 System.***out***.println(**"我应该是什么分类"**);  
 }  
 }  
  
 **public static void** main(String[] args) {  
 **new** CodeBlockDemo().test();  
 {  
 System.***out***.println(**"main"**);  
 }  
 }  
  
}

执行结果：

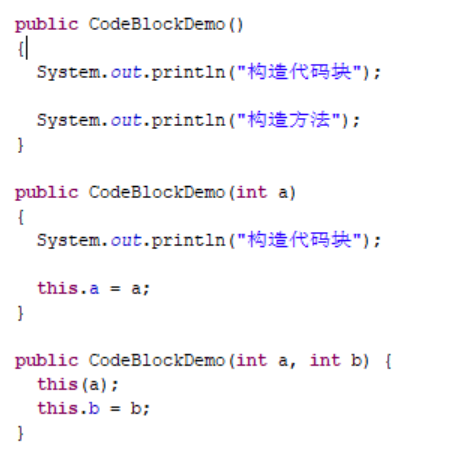
构造代码块

构造方法

test

我应该是什么分类

main

**public class** CodeBlockDemo {  
  
 **int a**;  
 **int b**;  
  
 **public** CodeBlockDemo(){ **反编译以后变成这样↓**  
 System.***out***.println(**"构造方法"**);  
 }  
  
 **public** CodeBlockDemo(**int** a){   
 **this**.**a** = a;  
 }

**//this(a)表示调用CodeBlockDemo(int a)**

**this()表示调用CodeBlockDemo()**

**这里都是表示调用构造方法** **public** CodeBlockDemo(**int** a,**int** b){  
 **this**(a);  
 **this**.**b** = b;  
 }  
  
 {  
 System.***out***.println(**"构造代码块"**);  
 }  
  
 **public void** test(){  
 System.***out***.println(**"test"**);  
 {  
 System.***out***.println(**"我应该是什么分类"**);  
 }  
 }  
  
 **public static void** main(String[] args) {  
 **new** CodeBlockDemo().test();  
 {  
 System.***out***.println(**"main"**);  
 }  
 }  
}

执行结果：

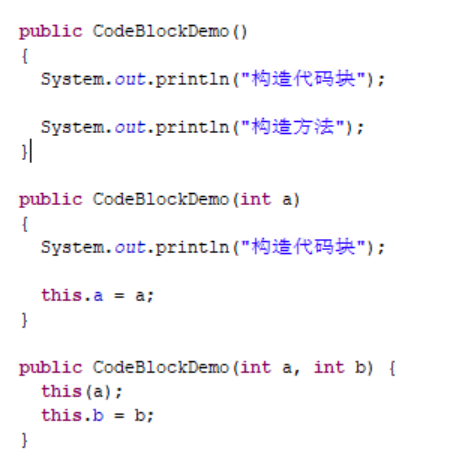
构造代码块

构造方法

test

我应该是什么分类

main

**public class** CodeBlockDemo {  
  
 **int a**;  
 **int b**;  
  
 **public** CodeBlockDemo(){  
  
 System.***out***.println(**"构造方法"**);  
 }  
  
 **public** CodeBlockDemo(**int** a){  
 **this**.**a** = a;  
 }  
  
 **public** CodeBlockDemo(**int** a,**int** b){  
 **this**(a);  
 **this**.**b** = b;  
 }  
  
  
 {  
 System.***out***.println(**"构造代码块"**);  
 }  
  
 **public void** test(){  
 System.***out***.println(**"test"**);  
 {  
 System.***out***.println(**"我应该是什么分类"**);  
 }  
 }  
  
  
 **public static void** main(String[] args) {  
 **new** CodeBlockDemo(1,2).test();  
 {  
 System.***out***.println(**"main"**);  
 }  
 }  
  
  
}

执行结果：

构造代码块

test

我应该是什么分类

main

**public class** CodeBlockDemo {  
  
 **int a**;  
 **int b**;  
  
 **public** CodeBlockDemo(){  
  
 System.***out***.println(**"构造方法"**);  
 }  
  
 **public** CodeBlockDemo(**int** a){  
 **this**.**a** = a;  
 }  
  
 **public** CodeBlockDemo(**int** a,**int** b){  
 **this**.**a** = a;  
 **this**.**b** = b;  
 }  
  
  
 {  
 System.***out***.println(**"构造代码块"**);  
 }  
  
 **public void** test(){  
 System.***out***.println(**"test"**);  
 {  
 System.***out***.println(**"我应该是什么分类"**);  
 }  
 }  
  
  
 **public static void** main(String[] args) {  
 **new** CodeBlockDemo(1,2).test();  
 {  
 System.***out***.println(**"main"**);  
 }  
 }  
}

执行方法:

构造代码块

test

我应该是什么分类

main

* 静态代码块→使用static声明的代码块

使用static{}括起来的代码叫做静态代码块，是在程序载入的时候优先执行，即在new对象之前就有它，所以它是首要执行的代码块，不管位置放在哪儿。即它时在类初始化时执行，不是在创建对象时执行

类第一次被载入时先执行static代码块；类多次载入时，static代码块只执行一次；Static经常用来进行static变量的初始化

数据库连接等其他提前需要准备好的代码会放在static代码块中

* 同步代码块→多线程的时候会学到

在多线程的时候会使用，用来给共享空间进行加锁操作

⭐这些代码块的执行顺序：

静态代码块→构造代码块（创建对象的时候才会用到）→普通代码块

**package（打包）**

* 为什么需要package？

对应到文件系统就是多级目录。为了解决类之间的重名问题。为了便于管理类，合适的类位于合适的包。将具体处理功能的代码放到同一个目录下

* package怎么用？

通常是类的第一句非注释性语句

格式：package 域名的倒写

如[www.mashibing.com](http://www.mashibing.com) → package com.mashibing.sql

包名：域名倒着写即可，再加上模块名，并与内部管理类

在IDE里创建package，比如写com.mashibing，就会创建出来com目录，mashibing为包含在com目录里的目录。这时在mashibing里新建类，在这个类的最上方会出现↓

**package** com.mashibing;

完全限定名：包名+类名

* 注意：

写项目时都要加包，不要使用默认包

com.gao和com.gao.car，这两个包没有包含关系，是两个完全独立的包。只是逻辑上看起来后者是前者的一部分

**JDK中的主要包：**

java.lang（不需要手动导入，自动加载）

包含一些Java语言的核心类，如String，Math，Integer，System和Thread，提供常用功能

java.awt

包含构成抽象窗口工具集(abstract window toolkits)的多个类，这些类被用来构建和管理应用程序的图形用户界面(GUI)

java.net

包含执行与网络相关的操作的类

java.io

包含能提供多种输入/输出功能的类

java.util

包含一些实用工具类，如定义系统特性，使用与日期日历相关的函数

**import**

* 为什么需要import？

如果不使用import，我们如果用到其他包的类时，只能这么写：

java.util.Date date = **new** java.util.Date();，代码量太大，不利于编写与维护。通过import可以导入其他包下面的类，从而可以在本类种直接通过类名来调用

* import怎么用？

import java.util.Date; 这样就可以这样写：Date date = **new** Date(); 推荐！！！

import java.util.**\***; //导入该包下所有的类。会降低编译速度，但不会降低运行速度

* 注意：

java会默认导入java.lang包下所有的类，因此这些类我们可以直接使用

如果导入两个同名的类，只能用包名+类名来显示调用相关类：

java.util.Date date = new java.util.Date();

当一个java文件中需要使用多个同名的类的时候，只能选择导入一个，另一个使用完全限定名的方式进行导入。比如我导入了java.util.Date包，这时要使用java.sql.Date的时候只能用java.sql.Date date = new java.sql.Date();

* 静态导包：

当需要使用某个类的多个方法的时候，同时又不想频繁写该类的名称，此时可以使用静态导包。用于导入指定类的静态属性

例如：没有使用静态导包之前，求一个数的开方和绝对值↓

System.***out***.println(Math.*sqrt*(2));  
System.***out***.println(Math.*abs*(-2));

在静态导包import static java.lang.Math.\*;以后↓

System.***out***.println(*sqrt*(2));  
System.***out***.println(*abs*(-2));