## Java中方法介绍

### 04.19\_Java语言基础(方法概述和格式说明)

A:为什么要有方法

提高代码的复用性和维护性.

B:什么是方法

完成特定功能的代码块。

**C:方法的格式**

**修饰符 返回值类型 方法名(参数类型 参数名1,参数类型 参数名2...) {**

**方法体语句;**

**return 返回值; //任何方法都是有返回值的,void不过就是一个return;而已**

**}**

D:方法的格式说明

修饰符：目前就用 public static。后面我们再详细的讲解其他的修饰符。

返回值类型：就是功能结果的数据类型。

方法名：符合命名规则即可。方便我们的调用。

参数：

实际参数：就是实际参与运算的，由外边传进来的。

形式参数；就是方法定义上的，用于接收实际参数的。形式参数和下边的方法体语句可能发生关系，也可能不方法生关系的，也可能只发生一个参数的关系。

参数类型：就是参数的数据类型。

参数名：就是变量名

方法体语句：就是完成功能的代码。

return：结束方法的。

返回值：就是功能的结果，由return带给调用者，记得要带回什么。

### 04.20\_Java语言基础(方法之求和案例及其调用)

A:如何写一个方法，写方法之前应该明确这两个参数的定义。

1,明确返回值类型

2,明确参数列表

B:案例演示

需求：求两个数据之和的案例

C:方法调用图解

int sum = add(10,20);

System.out.println(sum);

//add(30,40); //有返回值方法的单独调用,没有意义

System.out.println(add(30,40)); //这样调用是可以,but如果需要用这个结果不推荐这样调用

public static int add(int x,int y) { //形式参数应该一致。

int sum = x + y;

return sum; //如果有返回值必须用return语句带回

}

注意有不一样的：

import java.util.Scanner;

class Demo11\_Array {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.println("请输入对应的星期范围在1-7");

int week = sc.nextInt();

System.out.println("星期" + getWeek(week));

}

public static char getWeek(int week) { //int和char注意区别！

char[] arr = {' ','一','二','三','四','五','六','日'};

return arr[week];

}

}

### 04.21\_Java语言基础(方法的注意事项)

A:方法调用(有具体返值回)

a:单独调用,一般来说没有意义，所以不推荐。 eg：add(30,40)

b:输出调用,但是不够好。因为我们可能需要针对结果进行进一步的操作。System.out.println(add(30,40))

c:赋值调用,推荐方案。int z = add(30,40)

B:案例演示

a:方法不调用不执行

b:方法与方法是平级关系，不能嵌套定义

c:方法定义的时候形式参数之间用逗号隔开

d:方法调用的时候不用再传递数据类型，但是要符合定好的数据类型。

e:如果方法有明确的返回值，一定要有return带回一个值

### 04.22\_Java语言基础(方法的练习)

A:案例演示

\*\*需求：键盘录入两个数据，返回两个数中的较大值\*\*

B:案例演示

\*\*需求：键盘录入两个数据，比较两个数是否相等\*\*

import java.util.Scanner;

class Test1\_Method {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in); //创建键盘录入对象

System.out.println("请输入第一个整数:");

int x = sc.nextInt(); //将键盘录入的整数存储在x中

System.out.println("请输入第二个整数:");

int y = sc.nextInt(); //将键盘录入的整数存储在y中

//int max = getMax(x,y);

//System.out.println(max);

boolean b = isEquals(x,y);

System.out.println(b);

}

/\*

返回连个整数的较大值

1,明确返回值类型 int

2,明确参数列表 int a,int b

public static int getMax(int a,int b) {

return a > b ? a : b; //注意这里的写法

}

\*/

/\*

判断两个整数是否相等

1,明确返回值类型 boolean

2,明确参数列表 int a,int b

public static boolean isEquals(int a,int b) {

return a == b;

}

}

\*/

### 04.23\_Java语言基础(方法之输出星形及其调用)

A:案例演示

需求：根据键盘录入的行数和列数，在控制台输出星形

B:方法调用：(无返回值,void)

单独调用

输出调用(错误) 无法输出的

赋值调用(错误) 怎么接受，void根本就不是数据类型。

import java.util.Scanner;

class Demo3\_Method {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in); //创建键盘录入对象

System.out.println("请输入行数:");

int row = sc.nextInt(); //将键盘录入的行数存储在row中

System.out.println("请输入列数:");

int column = sc.nextInt(); //将键盘录入的列数存储在column中 //System.out.println(print(row,column)); //错误: 此处不允许使用 '空' 类型,返回值是void的方法不能输出调用

//返回值是void的方法只能单独调用

print(row,column);

}

/\*

在控制台输出矩形星形

1,明确返回值类型,经分析没有具体的返回值类型,void

2,明确参数列表int a,int b

\*/

public static void print(int a,int b) {

for (int i = 1;i <= a ;i++ ) { //行数

for (int j = 1;j <= b ;j++ ) { //列数

System.out.print("\*");

}

System.out.println();

}

//return ; //如果返回值类型是void,return可以省略,即使省略系统也会默认给加上,形式是return;

}

}

### 04.24\_Java语言基础(方法的练习)

A:案例演示

需求：根据键盘录入的数据输出对应的乘法表

import java.util.Scanner;

class Test2\_Method {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in); //创建键盘录入对象

System.out.println("请录入一个整数,范围在1-9之间");

int num = sc.nextInt(); //将键盘录入的整数存储在num中

print99(num);

}

/\*

打印99乘法表

1,返回值类型void

2,参数列表,int a

\*/

public static void print99(int a) {

for (int i = 1;i <= a ;i++ ) { //行数

for (int j = 1;j <= i ;j++ ) { //列数

System.out.print(j + "\*" + i + "=" + (i j) + "\t" );

}

System.out.println();

}

}

}

### 04.25\_Java语言基础(方法重载概述和基本使用)

A:方法重载概述

求和案例

2个整数

3个整数

4个整数

B:方法重载：

**在同一个类中，方法名相同，参数列表不同。与返回值类型无关。**

参数列表不同：

A:参数个数不同

B:参数类型不同

参数的顺序不同(算重载,但是在开发中不用)

class Demo4\_Overload { //overload重载

public static void main(String[] args) {

double sum1 = add(10,20.1);

System.out.println(sum1);

int sum2 = add(10,20,30);

System.out.println(sum2);

double sum3 = add(12.3,13);

System.out.println(sum3);

}

/\*

求两个整数的和

1,返回值类型int

2,参数列表 int a,int b

\*/

public static double add(int a,double b) {

return a + b;

}

/\*

求三个整数的和

1,返回值类型int

2,参数列表 int a,int b,int c

\*/

public static int add(int a,int b,int c) {

return a + b + c;

}

/\*

求两个小数的和

1,返回值类型double

2,参数列表 double a,double b

\*/

public static double add(double a,int b) {

return a + b;

}

}

已知函数 : void show(int a, int b, float c){ }，

哪个答案和show不是函数重载？ b

A.void show(int a,float c,int b){ }

B,void show(int x,int y,float z){ }

C.int show(int a, float c, int b){return a;}

D.int show(int a, float c ){return a;}

### 04.26\_Java语言基础(方法重载练习比较数据是否相等)

A:案例演示

需求：比较两个数据是否相等。

参数类型分别为两个int类型，两个double类型，并在main方法中进行测试

class Test3\_Overload {

public static void main(String[] args) {

boolean b1 = isEquals(10,10);

System.out.println(b1);

boolean b2 = isEquals(10.5,10.5);

System.out.println(b2);

}

/\*

比较两个数据是否相等

1,返回值类型boolean

2,参数列表int a,int b

\*/

public static boolean isEquals(int a,int b) {

return a == b;

}

/\*

比较两个数据是否相等

1,返回值类型boolean

2,参数列表double a,double b

\*/

public static boolean isEquals(double a,double b) {

return a == b;

}

}

### 04.27\_day04总结

把今天的知识点总结一遍。

## 面向对象

### 6.01\_面向对象(面向对象思想概述)(了解)

A:面向过程思想概述

      第一步

      第二步.........

举例：1.什么是面向过程和面向对象。（思想是根基很重要）

答： 什么是面向过程（c语言就是面向过程）？比如说，我们想要完成一个大型的项目，那想完成这个项目需要由几个功能共同实现，那在这里面我们就是以函数（具有特定功能的一段独立的小程序）为基础，用函数来完成各项操作，那强调就是函数，这就是面向过程。什么是面向对象？比如说，我们想要完成一个大型的项目，那想完成这个项目需要由几个功能共同实现，那我们换了一种方法，把这几个函数归类都扔到一个对象中，那这里面我们就是以对象为基础，用对象来完成各项操作，那强调就是对象，这就是面向对象。面向对象是以面向过程为基础的，是基于面向过程的。

书上总结的比较正确本人偏向于这一种：**面向过程就是分析解决问题的时候所需要的步骤，然后用函数来把这些步骤一一的实现，使用的时候一个一个的依次调用就好了。面向对象则是把解决的问题按照一定的规则划分为多个独立的对象，然后通过调用对象的方法来解决问题。**

B:面向对象思想概述

     找对象(第一步,第二步)

C:举例

     买煎饼果子

     洗衣服

D:面向对象思想特点

     a:是一种更符合我们思想习惯的思想，就是懒人思想

     b:可以将复杂的事情简单化

     c:将我们从执行者变成了指挥者

         角色发生了转换

E:面向对象开发

     就是不断的创建对象，使用对象，指挥对象做事情。

**F:面向对象设计**

**其实就是在管理和维护对象之间的关系。**

**G:面向对象特征**

**封装(encapsulation)**

**继承(inheritance)**

**多态(polymorphism)**

H:练习与问答

1、面向过程与面向对象有哪些区别与联系？最好能举例说明 a：先分别的叙述什么是面向对象什么是面向过程，然后再是区别和联系，举例建筑施工。

面向对象的思想特点有哪些？

面向对象的三大特征分别是什么？

### 6.02\_面向对象(类与对象概述)(掌握)

 A:我们学习编程是为了什么

     为了把我们日常生活中实物用学习语言描述出来

 B:我们如何描述现实世界事物

     属性    就是该事物的描述信息(事物身上的名词)事物的外在特征。成员变量

     行为    就是该事物能够做什么(事物身上的动词)事物所具有的能力、功能。成员方法

 C:Java中最基本的单位是类,Java中用class描述事物也是如此

     成员变量    就是事物的属性 比如人的年龄性别血型等等

     成员方法    就是事物的行为 比如人的吃喝拉撒睡

D:定义类其实就是定义类的成员(成员变量和成员方法)

a:成员变量

  和以前定义变量是一样的，只不过位置发生了改变。在类中，方法外。

b:成员方法

   和以前定义方法是一样的，只不过把static去掉，后面在详细讲解static的作用，也可以是static方法。

E:类和对象的概念

a:类：是一组相关的属性和行为的集合。eg：什么是类？

         官方:**事物的属性和行为的集合，是一个抽象的概念。**我们想要知道Java中的类是什么，就要先知道我们现实生活中的类是什么，因为现实生活中的类和Java中的类似是一样一样的，那现实生活中的类是什么？通过查看百度百科，许多相似或者相同事物的综合，具有共同特征的事物所形成的种类，比如说，我们称为人“类”，因为我们有共同的相似的特征，比如说，我们都有姓名，年龄，血型，星座等等，这些是我们的属性，我们都有吃喝拉撒睡的能力，这些是我们的行为，Java中的类就是这个类，只不过需要我们用代码体现出来

b:对象：是该类事物的具体体现

         官方:**对象类的具体存在**，体现。比如说你，我，他，我们都是活生生的个体，实体，实体就是对象

c:举例：

         类     学生这个群体

         对象    具体的某个学生就是一个对象（对象中包含了变量和方法）

练习与问答

1、什么是类，什么是对象？他们在生活中的体现分别是什么。

2、什么是成员变量，什么是成员方法？注意四种类，对象，成员变量，成员方法区别

### ////6.03\_面向对象(学生类的定义：先定义类名，然后是成员变量和成员方法！)

 A:学生事物

 B:学生类

 C:案例演示

     属性:姓名,年龄,性别

     行为:学习,睡觉

---------------------------------------------------------------------------------------------------------------------------

class Student {

String name; //姓名

int age; //年龄

String gender; //性别

public void study() { //定义学习的方法

System.out.println("学生学习");

}

public void sleep() { //定义睡觉的方法

System.out.println("学生睡觉");

}

}

---------------------------------------------------------------------------------------------------------------------------

### ////6.04\_面向对象(手机类的定义)(掌握)

 模仿学生类，让学生自己完成

     属性:品牌(brand)价格(price)

     行为:打电话(call),发信息(sendMessage)玩游戏(playGame)

---------------------------------------------------------------------------------------------------------------------------

class Phone {

String brand; //品牌

int price; //价格

public void call() { //打电话

System.out.println("打电话");

}

public void sendMessage() { //发信息

System.out.println("发信息");

}

public void playGame() { //玩游戏

System.out.println("玩游戏");

}

}

---------------------------------------------------------------------------------------------------------------------------

### ////6.05\_面向对象(学生类的使用)(掌握)

 A:文件名问题

**在一个java文件中必须写两个类：一个基本的类（就是对象的属性和方法），一个测试类（就是主函数，必须要有main函数，主要就是在主函数中调用的,为什么叫测试类函数，就是为了测试基本函数的！）（一般是在一个文件夹不是在一个文件中）。**

**建议：文件名称和测试类名称一致。**

**B:如何使用对象?**

**创建对象并使用，对象名就是合法的标识符，第一个小写以后都是大写。**

**格式：类名 对象名 = new 类名();    //注意类名是基本类的名称。(也可以是接口名，抽象类名？后边哪里有讲到的？注意这里)**

**D:如何使用**成员**变量呢?**

**对象名.变量名                  //某一个相关的数字，与下边的s一样。**

**对象名.变量名 = 数据 //定义对象成员属性的方式，未私有化的时候**

**对象名.变量名（参数） //函数已经变成了方法，是在向方法中传参。**

**E:如何使用**成员**方法呢?**

**对象名.方法名(...) //直接调用基本类的方法。****，有参数的话就传参。**

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_Student {

                public static void main(String[] args) {

                    //创建对象的格式:基本类名 对象名（某一个对象的名称） = new 基本类名();

                    //对象名:其实就是合法的标识符,如果是一个单词所有字母小写,如果是多个单词,从第二个单词开始首字母大写

                    Student s = new Student();//对象名这里是随便取的。

                    // D:如何使用成员变量呢?

                    // 对象名.变量名

                    s.name = "张三";

                    s.age = 23;

                    System.out.println(s.name + "..." + s.age);

                    // E:如何使用成员方法呢?

                    // 对象名.方法名(...)   //若是包含参数记得加入实际参数

                    s.study();

                    s.sleep();

                }

            }     //这个只是测试类函数，下边还应该写上基本类函数。

            class Student {

String name; //姓名

int age; //年龄

String gender; //性别

public void study() { //定义学习的方法

System.out.println("学生学习");

}

public void sleep() { //定义睡觉的方法

System.out.println("学生睡觉");

}

}

---------------------------------------------------------------------------------------------------------------------------

### ////6.06\_面向对象(手机类的使用)(掌握)

 A:学生自己完成

     模仿学生类，让学生自己完成

---------------------------------------------------------------------------------------------------------------------------

class Demo2\_Phone {

public static void main(String[] args) {

Phone p = new Phone(); //创建对象

p.brand = "锤子";

p.price = 998; //调用对象中的属性并赋值

System.out.println(p.brand + "..." + p.price); //调用的属性用p.brand，不用加括号。

p.call();

p.sendMessage();

p.playGame(); //调用成员方法

}

}

class Phone {

String brand; //品牌

int price; //价格

public void call() { //打电话

System.out.println("打电话");

}

public void sendMessage() { //发信息

System.out.println("发信息");

}

public void playGame() { //玩游戏

System.out.println("玩游戏");

}

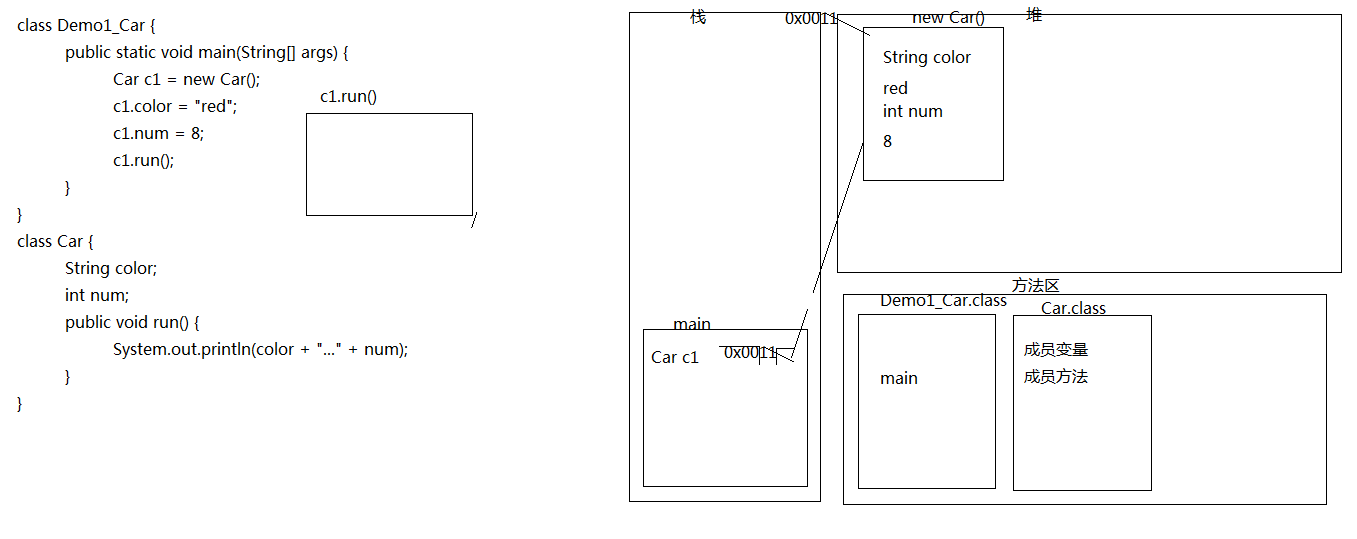
}

---------------------------------------------------------------------------------------------------------------------------

## 6.07\_面向对象(一个对象的内存图)(掌握)

 A:画图演示

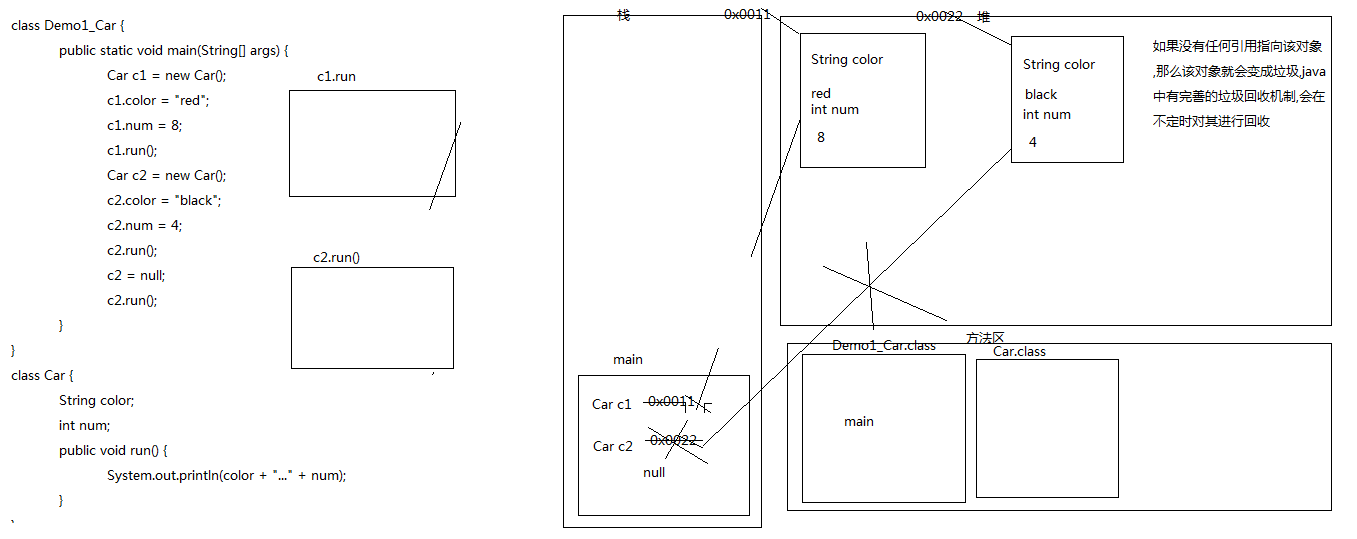
     一个对象



## 6.08\_面向对象(二个对象的内存图)(了解)

 A:画图演示

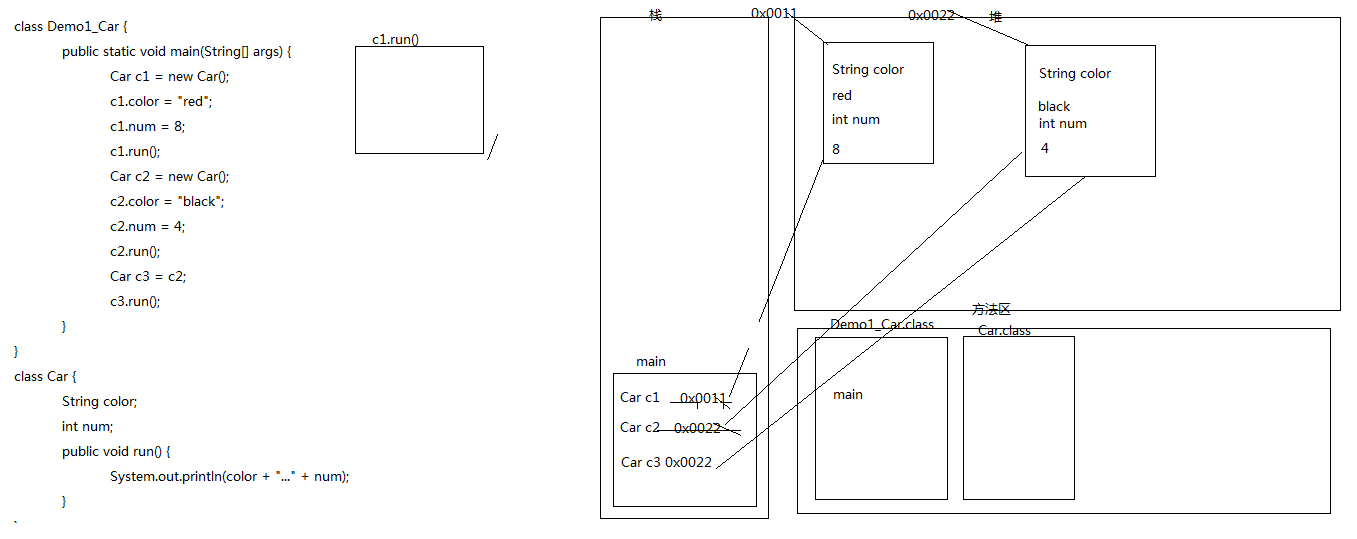
     二个不同的对象



## 6.09\_面向对象(三个引用两个对象的内存图)(了解)

 A:画图演示

     三个引用，有两个对象的引用指向同一个地址



## 6.10\_面向对象(成员变量和局部变量的区别)(掌握)

 A:在类中的位置不同

     成员变量：在类中方法外

     局部变量：在方法定义中或者方法声明上

 B:在内存中的位置不同

     成员变量：在堆内存(成员变量属于对象,对象进堆内存)

     局部变量：在栈内存(局部变量属于方法,方法进栈内存)

 C:生命周期不同

     成员变量：随着对象的创建而存在，随着对象的消失而消失

     局部变量：随着方法的调用而存在，随着方法的调用完毕而消失

 D:初始化值不同

     成员变量：有默认初始化值（基本数据类型的默认值）

     局部变量：没有默认初始化值，必须定义，赋值，然后才能使用。

**注意事项：**

**局部变量名称可以和成员变量名称一样**(不是在同一个地方所以可以的。**在方法中使用的时候，采用的是就近原则**（方法中如果我自己有定义变量我就用我自己的，没有的话我才用成员变量）。

**基本数据类型变量包些:byte,short,int,long,float,double,boolean,char。**

**引用数据类型变量包括哪些:数组,类,接口,枚举**

---------------------------------------------------------------------------------------------------------------------------

class Demo2\_Person {

public static void main(String[] args) {

Person p = new Person();

p.speak(); //调用成员方法，按照下边的会打印出来null和10.

}

}

class Person {

String name; //成员变量，默认值同数组的默认值。

int num;

public void speak() { //public void speak(int x)其中的x也是局部变量

int num = 10; //x和num都是局部变量

System.out.println(name); //即使不定义也会打印默认值。

System.out.println(num); //num就近原则所以打印的肯定是10。若是里边没有定义，就会是成员变量的默认值0.但是若是在成员方法中定义了但是没有初始化值，那么就会变异报错的。

}

}

---------------------------------------------------------------------------------------------------------------------------

## ////6.11\_面向对象(方法的形式参数是类名的时候如何调用)(掌握)

 A:方法的参数是类名

public void print(Student s){}//print(new Student());

**如果你看到了一个方法的形式参数是一个类类型(引用类型)，这里其实需要的是该类的对象**。

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_Student {

public static void main(String[] args) {

print(10);

Student s = new Student(); **//创建对象,并将对象的地址值赋值给s**

print(s);

}

public static void print(int x) { **//基本数据类型当作形式参数**

System.out.println(x);

}

**主函数中方法都加静态！非主函数均不加**

**public static void print(Student stu) { //引用数据类型当作形式参数**

**stu.name = "张三"; //注意这里的Student是一个类**

**stu.age = 23; //stu需要的是地址值，只有创建的学生对象才能传参**

**stu.speak(); //此时又调用了基本类的成员方法，底下是公开的所以能访问。方法是能调用方法的！**

}

}

class Student {

String name; //姓名

int age; //年龄

public void speak() {

System.out.println(name + "..." + age);

}

}

---------------------------------------------------------------------------------------------------------------------------

## 6.12\_面向对象(匿名对象的概述和应用)(掌握)

 A:什么是匿名对象

     没有名字的对象

**匿名对象的格式？**

**new 类名()，        eg：new Person()。**

 B:匿名对象应用场景

 a:调用方法，仅仅只调用一次的时候。

**new Person().name;   使用的是变量**

**new Person().eat();    使用的是方法**

         那么，这种匿名调用有什么好处吗?

             节省代码 ，源代码就是Person s = new Person();s.name = xxxx;

         注意：调用多次的时候，不适合。匿名对象调用完毕就是垃圾。可以被垃圾回收器回收。

**b:匿名**对象**可以作为实际参数传递。**

C:案例演示

     匿名对象应用场景，注意每一次的new就会增加内存的，注意内存图的画法。

---------------------------------------------------------------------------------------------------------------------------

class Demo2\_Car {

public static void main(String[] args) {

/\*Car c1 = new Car(); //创建有名字的对象

c1.run();

c1.run(); //第二次运行复制就行了

new Car().run(); //匿名对象调用方法，第二次就又创建了一次对象

new Car().run(); \*/ //匿名对象只适合对方法的一次调用,因为调用多次就会产生多个对象,不如用有名字的对象。**匿名对象是否可以调用属性并赋值?有什么意义?匿名对象可以调用属性,但是没有意义,因为调用后就变成垃圾如果需要赋值还是用有名字对象**

new Car().color = "red";

new Car().num = 8;

new Car().run();

}

}

class Car {

String color; //颜色

int num; //轮胎数

public void run() {

System.out.println(color + "..." + num);

}

}

---------------------------------------------------------------------------------------------------------------------------

class Demo3\_Car {

public static void main(String[] args) {

//Car c1 = new Car();

/\*c1.color = "red";

c1.num = 8;

c1.run();\*/

//method(c1);

method(new Car());

//Car c2 = new Car();

//method(c2);

method(new Car()); **//匿名对象可以当作参数传递**

}

//抽取方法提高代码的复用性

public static void method(Car cc) { //Car cc = new Car();

cc.color = "red";

cc.num = 8;

cc.run();

}

}

class Car {

String color; //颜色

int num; //轮胎数

public void run() {

System.out.println(color + "..." + num);

}

}

---------------------------------------------------------------------------------------------------------------------------

## 6.13\_面向对象(封装的概述)(掌握)

 A:封装概述

     是指隐藏对象的属性和实现细节（通过private修饰），仅对外提供公共访问方式。

 B:封装好处

     隐藏实现细节，提供公共的访问方式

     提高了代码的复用性

     提高安全性。

 C:封装原则

     将不需要对外提供的内容都隐藏起来。

     把属性隐藏，提供公共方法对其访问。

## 6.14\_面向对象(private关键字的概述和特点)(掌握)

 A:人类赋值年龄的问题

通过案例的演示说明对于年龄输入的限制。**我封装了，你要使用我就得符合我的内部条件。**

 B:private关键字特点

     a:是一个权限修饰符

**b:可以修饰成员变量和成员方法，还可以私有构造方法以免外部创建对象。**

**c:被其修饰的成员只能在本类中被访问，不可以再其他类中访问。在其他的类中需要通过getset的方式调用。（私有构造方法就不能再创建对象向了，也不能在方法的前边添加static来修饰，两者不共存的。若是私有构造方法可以通过再建立一个未被私有的方法，然后再去调用这个未被私有的就可以了，就是绕了一圈回到了终点，比较安全，这是一种思想）**

     d：怎么提供公共的访问方式？在有private修饰的成员变量的类中，写两个方法，在这两个方法中可以调用本类中的私有的成员变量，然后我们在测试类中创建该类对象，用对象调用这两个公共的方法，即可间接的实现调用私有成员变量。

 C:案例演示

     封装和private的应用：

     A:把成员变量用private修饰

     B:提供对应的getXxx()和setXxx()方法

     private仅仅是封装的一种体现形式,不能说封装就是私有。注:private仅仅是封装的一种体现，类和方法也是一种封装。

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_Person {

public static void main(String[] args) {

Person p1 = new Person();

p1.name = "张三"; //调用姓名属性并赋值

//p1.age = -17; //调用年龄属性并赋值

//p1.speak(); //调用行为

p1.setAge(-17); //其它类必须通过getset方法获取！

System.out.println(p1.getAge());

}

}

class Person {

String name; //姓名

private int age; //年龄

public void setAge(int a) { //设置年龄

if (a > 0 && a < 200) {

age = a;

}else {

System.out.println("请回火星吧,地球不适合你");

}

}

public int getAge() { //获取年龄 注意格式，是什么类型就返回什么类型

return age;

}

public void speak() {

System.out.println(name + "..." + age);

}

---------------------------------------------------------------------------------------------------------------------------

## 6.15\_面向对象(this关键字的概述和应用)(掌握)

 A:this关键字特点

**代表当前对象的引用**

 B:案例演示

**this的应用场景**

**用来区分成员变量和局部变量重名，谁人调用它他就代表谁。**

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_This {

public static void main(String[] args) {

Person p1 = new Person();

p1.setName("张三");

p1.setAge(23);

System.out.println(p1.getName() + "..." + p1.getAge());

Person p2 = new Person();

p2.setName("李四");

p2.setAge(24);

System.out.println(p2.getName() + "..." + p2.getAge());

}

}

class Person {

private String name; //姓名

private int age; //年龄

public void setAge(int age) { //设置年龄

if (age > 0 && age < 200) {

this.age = age; **//p1对象（地址）调用的时候就是代表p1**.

}else {

System.out.println("请回火星吧,地球不适合你");

}

}

public int getAge() { //获取年龄

return age;

}

public void setName(String name) { //设置姓名

this.name = name;

//System.out.println(name);

}

public String getName() { //获取姓名

return name;

}

}

---------------------------------------------------------------------------------------------------------------------------

### ////06.16\_面向对象(手机类代码及其测试)(掌握)

 A:学生练习

     请把手机类写成一个标准类，然后创建对象测试功能。

---------------------------------------------------------------------------------------------------------------------------

class Demo2\_Phone {

public static void main(String[] args) {

Phone p1 = new Phone();

p1.setBrand("三星");

p1.setPrice(5288);

System.out.println(p1.getBrand() + "..." + p1.getPrice());

p1.call();

p1.sendMessage();

p1.playGame();

}

}

class Phone { //java bean

private String brand; //品牌

private int price; //价格

public void setBrand(String brand) { //设置品牌

this.brand = brand;

}

public String getBrand() { //获取品牌

return this.brand; //this.可以省略,你不加系统会默认给你加

}

public void setPrice(int price) { //设置价格

this.price = price;

}

public int getPrice() { //获取价格

return price;

}

public void call() { //打电话

System.out.println("打电话");

}

public void sendMessage() { //发短信

System.out.println("发短信");

}

public void playGame() { //玩游戏

System.out.println("玩游戏");

}

}

---------------------------------------------------------------------------------------------------------------------------

06.17\_day06总结

 把今天的知识点总结一遍。

DAY07 构造方法，static，静态变量，main，说明书jdk，Math中Random

## 构造方法

### 7.01\_面向对象(构造方法Constructor概述和格式)(掌握)

A:构造方法概述和作用

给对象的数据(属性)进行初始化，就是对象一创建出来就已经具备了参数。

B:构造方法格式特点

a:方法名与类名相同(大小也要与类名一致)。

b:没有返回值类型，连void都没有

c:没有具体的返回值（就是return;就是什么也不返回。）

C:构造方法的格式：注意有空参和有参的区别。下边的只是有参的构造方式。（注意若是被private修饰的话就在类的外部无法创建对象了，只能通过类名.方法名来调用方法。而对象则是永远都不能被进行创建的了！）

修饰符 类名() {

return;

}

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_Constructor { //Constructor构造

public static void main(String[] args) {

Person p = new Person(); //在一创建对象的时候,系统就帮我调用了构造方法,后边的空括号就是在调用下边的构造方法。

//p.Person(); //构造方法不能被对象调用，因为是构造方法创造的对象，儿子不能调用父亲！

p.show(); //对象能调用成员方法，因为这是他自身的行为。

Person p2 = new Person(); //再次创建对象，也会按照原来的输出,没有发生重载，后边将讲到重载。

p2.show();

}

}

class Person {

private String name;

private int age;

public Person() { //构造方法

//System.out.println("Hello World!"); //这两句只是为了演示，根本就不用写的。

//return; //构造方法也是有return语句的,格式是return;和void一样。不写系统默认有，但是被省略了。

name = "张三"; //空参可以没有this，但是有参构造必须要有this

age = 23;

}

public void show() { //在定义show的方法，是在class类中，所以可以直接写name和age。

System.out.println(name + "..." + age);

}

}

---------------------------------------------------------------------------------------------------------------------------

### 7.02\_面向对象(构造方法的重载及注意事项)(掌握)

A:案例演示

**构造方法的重载**

重载:方法名相同（**在这里就是类名（也是方法名，两者是一模一样的），类名肯定是一样的了，若是不一样就不能放在同一个类下**）,与返回值类型无关**(构造方法根本就没有返回值)**,只看参数列表，总结：**在这里构造方法造成重载就只有参数的顺序不同（但是开发中不这么用）和参数个数不同了。**

B:构造方法注意事项

a:如果我们没有给出构造方法，系统将自动提供一个无参构造方法。

b:**如果我们给出了构造方法，系统将不再提供默认的无参构造方法**。

注意：这个时候，如果我们还想使用无参构造方法，就必须自己给出。**建议永远自己给出无参构造方法**

C：因为构造也是方法，既然是方法，那么就要满足重载的条件。

当一类中存在多个构造方法的时候就应该是重载。重载就是参数列表不同，即类型和个数不同（顺序不同也是），与返回值无关。

---------------------------------------------------------------------------------------------------------------------------

class Demo2\_Person {

public static void main(String[] args) {

Person p1 = new Person(); //这是空参的创建

p1.show(); //day06所学的基本类方法的调用。

System.out.println("---------------------");

Person p2 = new Person("张三",23); //注意和下边的p3是同一种，并不是重载。只是天生的传参不同罢了。

p2.show();

System.out.println("---------------------");

Person p3 = new Person("李四",24);

p3.show();

}

}

class Person {

private String name; //姓名

private int age; //年龄

public Person() { //空参构造

System.out.println("空参的构造");

}

public Person(String name,int age) { //有参的构造创建对象的时候应该是进行有参的创建。

this.name = name;

this.age = age;

//无参的创建此时系统已经不会自动的给我们创建了，我们应该自己手动的创建上。

System.out.println("有参的构造"); //这句话就是一个演示，应该省略掉，重要的是上边的代码。

}

public void show() { //主要是为了展示上边的赋值情况

System.out.println(name + "..." + age);

}

}

---------------------------------------------------------------------------------------------------------------------------

## 成员变量

### 7.03\_面向对象(给成员变量赋值的两种方式的区别)

A:setXxx()方法

**修改属性值,比较灵活，常用的方法。开发中很常用的。**

B:构造方法

给对象中属性进行初始化，天生来就有的，后天不能改变，改那也是新的对象了。

---------------------------------------------------------------------------------------------------------------------------

class Demo3\_Person {

public static void main(String[] args) {

Person p1 = new Person("张三",23);

//p1 = new Person("张天一",23); //这种方式看运行结果貌似是改名了,其实是将原对象变成垃圾。因为new了。

System.out.println(p1.getName() + "..." + p1.getAge()); System.out.println("--------------------");

Person p2 = new Person(); //空参构造创建对象

**p2.setName("李四"); //注意这个时候setName已经是方法了没有被私有的时候是参数可以赋值，用等号。现在是方法只能是带参数的传参的！。**

p2.setAge(24);

p2.setName("李鬼"); //这才是真的再改名字,没有创建新的对象。

System.out.println(p2.getName() + "..." + p2.getAge()); **//p2.getName调用属性，加括号是调用方法。**

}

}

/\*构造方法

给属性进行初始化.(有参不好改名字，像上边一样那是在创建新的对象。）

setXxx方法

修改属性值

这两种方式,在开发中用setXxx更多一些,因为比较灵活\*/

class Person {

private String name; //姓名

private int age; //年龄

**public Person() {} //空参构造**

**public Person(String name,int age) { //有参构造**

**this.name = name;**

**this.age = age;**

**}**

**public void setName(String name) { //设置姓名**

**this.name = name;**

**}**

**public String getName() { //获取姓名**

**return name;**

**}**

public void setAge(int age) { //设置年龄

this.age = age;

}

public int getAge() { //获取年龄

return age;

}

}

---------------------------------------------------------------------------------------------------------------------------

### ////7.04\_面向对象(学生类的代码及测试)(掌握)

A:案例演示

学生类：

成员变量：

name，age

构造方法：

无参，带两个参

成员方法：

getXxx()/setXxx()

show()：输出该类的所有成员变量值

B:给成员变量赋值：

a:setXxx()方法，开发中常用的，比较灵活的。

b:构造方法

C:输出成员变量值的方式：（注意一下的两种方式输出）

a:通过getXxx()分别获取然后拼接

b:通过调用show()方法搞定

---------------------------------------------------------------------------------------------------------------------------

class Demo4\_Student {

public static void main(String[] args) {

Student s1 = new Student(); //使用空参构造

s1.setName("张三"); //设置姓名

s1.setAge(23); //设置年龄

//注意这里是赋值，然后系统才能返回值 s1.setName();但是不能直接的输出，需要引用，如下变所示：

System.out.println("我的姓名是:" + s1.getName() + ",我的年龄是:" + s1.getAge());

//getXxx()获取属性值,可以打印,也可以赋值给其他的变量,做其他的操作

Student s2 = new Student("李四",24);

s2.show(); //只是为了显示属性值，天生的所以才能调用 show 方法。

}

}

class Student {

private String name; //姓名

private int age; //年龄

public Student(){} //空参构造

public Student(String name,int age) { //有参构造

this.name = name;

this.age = age;

}

public void setName(String name) { //设置姓名

this.name = name;

}

public String getName() { //获取姓名

return name;

}

public void setAge(int age) { //设置年龄

this.age = age;

}

public int getAge() { //获取年龄

return age;

}

public void show() {

System.out.println("我的姓名是:" + name + ",我的年龄是:" + age);

} //只是为了展示方便而已，也可以是上边的无参输出方式。重点是在参数的上边不是在显示的上边。

} //注意这是在private的内部多以才可以使用name和age，在外部的话就只能是使用getName的方法了。

---------------------------------------------------------------------------------------------------------------------------

### ////7.05\_面向对象(手机类的代码及测试)(掌握)

A:案例演示

模仿学生类，完成手机类代码

---------------------------------------------------------------------------------------------------------------------------

class Demo5\_Phone {

public static void main(String[] args) {

Phone p1 = new Phone();

p1.setBrand("苹果");

p1.setPrice(1500);

System.out.println(p1.getBrand() + "..." + p1.getPrice());

Phone p2 = new Phone("小米",98);

p2.show();

}

}

class Phone {

private String brand; //品牌

private int price; //价格

public Phone(){} //空参构造

public Phone(String brand,int price) { //有参构造

this.brand = brand;

this.price = price;

}

public void setBrand(String brand) { //设置品牌

this.brand = brand;

}

public String getBrand() { //获取品牌

return brand;

}

public void setPrice(int price) { //设置价格

this.price = price;

}

public int getPrice() { //获取价格

return price;

}

public void show() {

System.out.println(brand + "..." + price);

}

}

---------------------------------------------------------------------------------------------------------------------------

### 7.06\_面向对象(创建一个对象的步骤)(掌握)

A:画图演示

画图说明一个对象的创建过程做了哪些事情?

Student s = new Student();

1,Student.class加载进内存

2,声明一个Student类型引用s

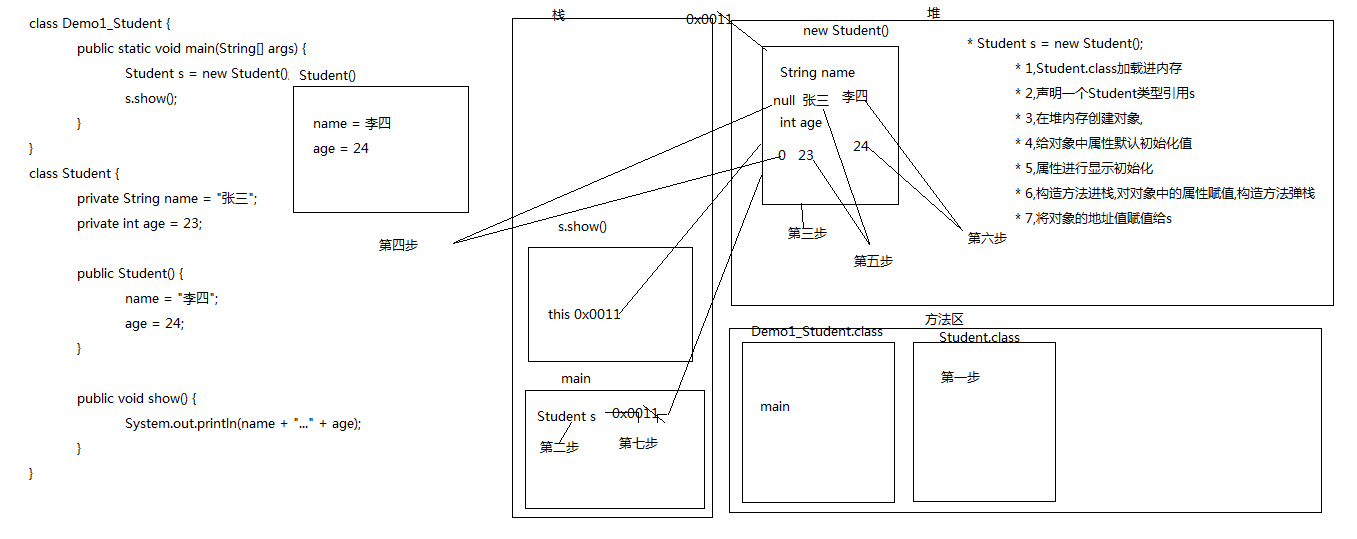
3,在堆内存创建对象,

4,给对象中属性默认初始化值

5,属性进行显示初始化

6,构造方法进栈,对对象中的属性赋值,构造方法弹栈

7,将对象的地址值赋值给s。



### ////7.07\_面向对象(长方形案例练习)(掌握)

A:案例演示

需求：

定义一个长方形类,定义 求周长和面积的方法，

然后定义一个测试类进行测试。

---------------------------------------------------------------------------------------------------------------------------

class Test1\_Rectangle { //Rectangle矩形

public static void main(String[] args) {

Rectangle r = new Rectangle(10,20);

System.out.println(r.getLength()); //周长

System.out.println(r.getArea()); //面积

}

}

class Rectangle {

private int width; //宽

private int high; //高

public Rectangle(){} //空参构造

public Rectangle(int width,int high) {

this.width = width; //有参构造

this.high = high;

}

public void setWidth(int width) {//设置宽

this.width = width;

}

public int getWidth() { //获取宽

return width;

}

public void setHigh(int high) { //设置高

this.high = high;

}

public int getHigh() { //获取高

return high;

}

public int getLength() { //获取周长 这是可以的，只有set方法或者只有get方法，不是一一对应的，但是前后是有照顾到的。

return 2 (width + high);

}

public int getArea() { //获取面积

return width high;

}

}

---------------------------------------------------------------------------------------------------------------------------

### ////7.08\_面向对象(员工类案例练习)(掌握)

A:案例演示

需求：定义一个员工类Employee

自己分析出几个成员，然后给出成员变量

姓名name,工号id,工资salary

构造方法，

空参和有参的

getXxx()setXxx()方法，

以及一个显示所有成员信息的方法。并测试。

---------------------------------------------------------------------------------------------------------------------------

class Test2\_Employee { //employee员工

public static void main(String[] args) {

Employee e = new Employee("令狐冲","9527",20000);

e.work();

}

}

class Employee {

private String name; //姓名

private String id; //工号

private double salary; //工资

public Employee() {} //空参构造

public Employee(String name, String id, double salary) {//有参构造

this.name = name;

this.id = id;

this.salary = salary;

}

public void setName(String name) { //设置姓名

this.name = name;

}

public String getName() { //获取姓名

return name;

}

public void setId(String id) { //设置id

this.id = id;

}

public String getId() { //获取id

return id;

}

public void setSalary(double salary) { //设置工资

this.salary = salary;

}

public double getSalary() { //获取工资

return salary;

}

public void work() {

System.out.println("我的姓名是:" + name + ",我的工号是:" + id + ",我的工资是:" + salary

+ ",我的工作内容是敲代码");

}

}

---------------------------------------------------------------------------------------------------------------------------

## Static关键字

### 7.09\_面向对象(static关键字及内存图)

A:案例演示

通过一个案例引入static关键字。

人类：Person。每个人都有国籍，中国。

B:画图演示

带有static的内存图

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_Static { //重点的是内存图的画法。

public static void main(String[] args) {

/\*Person p1 = new Person(); //创建对象

p1.name = "苍老师"; //调用姓名属性并赋值

p1.country = "日本"; //调用国籍属性并赋值,一个赋值后边的共享！

Person p2 = new Person();

p2.name = "小泽老师"; //调用姓名属性并赋值

//p2.country = "日本"; //调用国籍属性并赋值

p1.speak();

p2.speak();\*/

Person.country = "日本"; //静态多了一种调用方式,可以通过类名.

System.out.println(Person.country);

}

}

class Person {

String name; //姓名

static String country; //国籍

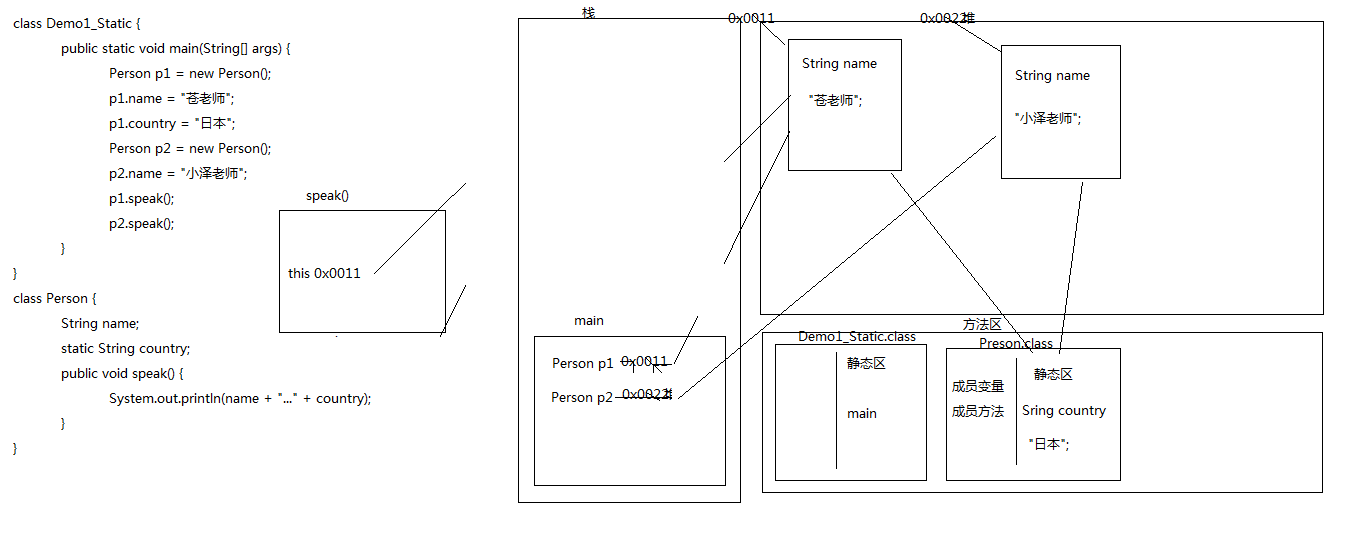
public void speak() { //说话的方法

System.out.println(name + "..." + country);

}

}

---------------------------------------------------------------------------------------------------------------------------



### 7.10\_面向对象(static关键字的特点)(掌握)

**A:static**关键**字的特点**

**a:随着类的加载而加载，static就是在类的下边。**

**b:优先于对象存在。这是公共使用的所以最先创建的。**

**c:被类的所有对象共享**

举例：咱们班级的学生应该共用同一个班级编号。

其实这个特点也是在告诉我们什么时候使用静态?

如果某个成员变量是被所有对象共享的，那么它就应该定义为静态的。

举例：

饮水机(用静态修饰)

水杯(不能用静态修饰)

**总结一句话：共性用静态,特性用非静态**

**d:可以通过类名调用**

其实它本身也可以通过对象名调用。

推荐使用类名调用，不用创建对象的，对象名调用也是没有问题的只是增加了内存，所以说还是这个好一些的。

静态修饰的内容一般我们称其为：与类相关的，类成员。静态就被称作是类静态。

B:案例演示

static关键字的特点

### 7.11\_面向对象(static的注意事项)(掌握)

A:static的注意事项

**a:在静态方法中是没有this关键字的**

如何理解呢?

静态是随着类的加载而加载，this是随着对象的创建而存在。

静态比对象先存在。

**b:静态方法只能访问静态的成员变量和静态的成员方法，构造性参数不能被修饰。**

**静态方法：**

成员变量：只能访问静态变量

成员方法：只能访问静态成员方法

非静态方法：

成员变量：可以是静态的，也可以是非静态的

成员方法：可是是静态的成员方法，也可以是非静态的成员方法。

简单记：

**静态只能访问静态。**

**在类的外部非静态的方法只能通过建立对象，然后通过对象.方法名去调用。在类内部直接调用即可。静态的方法在外部可以通过类名.方法名调用。这就理所应当的说明了在JDK的帮助文档中全部是static类型的，我们都是可以利用的，只是要加上类名。**

**是不需要创建对象的，同时内部全是静态就把构造方法私有了，就不让人创建对象了，免得别人再麻烦，同时增加了内存的消耗。**

**类加载并不是创建对象，只是包含了非静态区里边的成员变量和成员方法，没有创建！**

B:案例演示

static的注意事项

---------------------------------------------------------------------------------------------------------------------------

class Demo2\_Static {

public static void main(String[] args) {

//Demo d = new Demo();

//d.print1();

Demo.print2(); //调用静态成员的方法。

}

}

class Demo {

int num1 = 10; //非静态的成员变量

static int num2 = 20; //静态的成员变量

/\*public void print1() { **//非静态的成员方法,既可以访问静态的成员也可以访问非静态的**

System.out.println(num1);

System.out.println(num2);

}\*/

public static void print2() { //静态的成员方法

//System.out.println(**this.num1**);**//静态的成员方法不能访问非静态的,错误: 无法从静态上下文中引用非静态 变量 num1**

System.out.println(num2);

}

}

---------------------------------------------------------------------------------------------------------------------------

### 7.12\_面向对象(静态变量和成员变量的区别)(掌握)

**静态变量也叫类变量 成员变量也叫对象变量**

A:所属不同

静态变量属于类，所以也称为为类变量

成员变量属于对象，所以也称为实例变量(对象变量)

B:内存中位置不同

静态变量存储于方法区的静态区

成员变量存储于堆内存

C:内存出现时间不同

静态变量随着类的加载而加载，随着类的消失而消失

成员变量随着对象的创建而存在，随着对象的消失而消失

D:调用不同

静态变量可以通过类名调用，也可以通过对象调用

成员变量只能通过对象名调用

### ////7.13\_面向对象(main方法的格式详细解释)(了解)

A:格式

public static void main(String[] args) {}

B:针对格式的解释

public 被jvm调用，访问权限足够大。

static 被jvm调用，不用创建对象，直接类名访问，因为是静态的所以是共享的谁都可以访问到的。，若是非静态得先创建对象才可以访问。

void被jvm调用，不需要给jvm返回值

main 一个通用的名称，虽然不是关键字，但是被jvm识别

String[] args 以前用于接收键盘录入的，args是一个数组的名称，可以更改。

总结：其实这也是一种方法，被JVM类所调用了，不同的类肯定之间得静态访问。jvm是软件本身肯定就不需要返回值的，main只是一个主要的函数，是jvm调用的入口，然后通过面函数再访问其他的函数。

C:演示案例

通过args接收键盘例如数据，现在一般都不用了。

---------------------------------------------------------------------------------------------------------------------------

class Demo3\_Main {

public static void main(String[] args) {

System.out.println(args.length);

for (int i = 0;i < args.length ;i++ ) {

System.out.println(args[i]);

}

}

}

//定义一个静态的函数

class person {

public static void sayHello(){ //主函数的代表就是main，后边的录入系统已经不用了。

System.out.println("hello");

}

}

---------------------------------------------------------------------------------------------------------------------------

## 其他

### 7.14\_面向对象(工具类中使用静态)(了解)

A:制作一个工具类

ArrayTool

1,获取最大值

2,数组的遍历

3,数组的反转

**内部的方法都是静态的方法，就可以私有构造方法外部类不能创建对象了，爽！**

### 7.15\_面向对象(说明书的制作过程)(了解)

A:对工具类加入文档注释

B:通过javadoc命令生成说明书

@author(提取作者内容)

@version(提取版本内容)

这是在最前边的写的。亦可以写上其他的说明。

javadoc -d 指定的文件目录 -author -version ArrayTool.java

目录就是当前文件夹下（用点表示），版本与作者可以省略。

@param 参数名称 形式参数的数据类型

Eg：@param arr int(人性化点写上：接收一个int类型的)

@return 函数运行完返回的数据（没有返回值就不用在写了）

Eg：@return 返回最大值（max，见名知意）

---------------------------------------------------------------------------------------------------------------------------

/\*\*

这是一个数组工具类,里面封装了查找数组最大值,打印数组,数组反转的方法

@author fengjia

@version v1.0

\*/

public class ArrayTool {

**//如果一个类中所有的方法都是静态的,需要再多做一步,私有构造方法,目的是不让其他类创建本类对象**

**//直接用类名.调用即可**

**/\*\***

**私有构造方法 //这在说明书中根本就不体现。没有意义。**

**\*/**

**private ArrayTool(){}**

//1,获取最大值

/\*\*

这是获取数组中最大值的方法

@param arr 接收一个int类型数组

@return 返回数组中最大值

\*/

public **static** int getMax(int[] arr) {

int max = arr[0]; //记录第一个元素

for (int i = 1;i < arr.length ;i++ ) { //从第二个元素开始遍历

if (max < arr[i]) { //max与数组中其他的元素比较

max = arr[i]; //记录住较大的

}

}

return max; //将最大值返回

}

//2,数组的遍历

/\*\*

这是遍历数组的方法

@param arr 接收一个int类型数组

\*/

public static void print(int[] arr) {

for (int i = 0;i < arr.length ;i++ ) { //遍历数组

System.out.print(arr[i] + " ");

}

}

//3,数组的反转

/\*\*

这是数组反转的方法

@param arr 接收一个int类型数组

\*/

public static void revArray(int[] arr) {

for (int i = 0;i < arr.length / 2 ;i++ ) { //循环次数是元素个数的一半

/\*

arr[0]与arr[arr.length-1-0] 交换

arr[1]与arr[arr.length-1-1] 交换

arr[2]与arr[arr.length-1-2] 交换

\*/

int temp = arr[i];

arr[i] = arr[arr.length-1-i];

arr[arr.length-1-i] = temp;

}

}

}

class Demo1\_ArrayTool {

public static void main(String[] args) {

int[] arr = {33,11,22,66,55,44};

/\*ArrayTool at = new ArrayTool();

int max = at.getMax(arr); //获取最值

System.out.println(max);

System.out.println("---------------");

at.print(arr); //打印

System.out.println();

System.out.println("---------------");

System.out.println("反转后:");

at.revArray(arr); **//反转，反转后数组已经变化。下边就可以直接遍历**

at.print(arr); \*/

ArrayTool.print(arr);

}

}

---------------------------------------------------------------------------------------------------------------------------

### 7.16\_面向对象(如何使用JDK提供的帮助文档，所有的类都是static修饰的)(了解)

A:找到文档，打开文档

B:点击显示，找到索引，出现输入框

C:你应该知道你找谁?举例：Scanner，找的是类而不是方法!

D:看这个类的结构(需不需要导包)

成员变量 字段

构造方法 构造方法

成员方法 方法

## Math类

### 7.17\_面向对象(学习Math类的随机数功能)

打开JDK提供的帮助文档学习

A:Math类概述

类包含用于执行基本数学运算的方法

B:Math类特点

由于Math类在java.lang包下，所以不需要导包。Lang包是在java下的src文件夹下的。用editplus打开，记事本是乱码文件。

因为它的成员全部是静态的,所以私有了构造方法

C:获取随机数的方法

public static double random():返回带正号的 double 值，该值大于等于 0.0 且小于 1.0。

D:我要获取一个1-100之间的随机数

**\*\*int number = (int)(Math.random()\*100)+1;**

DAY08 代码块，extends的利弊，注意事项，成员变量关系，this和super，构造方法的访问，重写

## 代码块

### 8.01\_面向对象(代码块的概述和分类)(面试的时候会问,开发很少用)

A:代码块概述

在Java中，使用{}括起来的代码被称为代码块。

B:代码块分类

根据其位置和声明的不同，可以分为局部代码块，构造代码块，静态代码块，同步代码块(多线程讲解)。

C:常见代码块的应用

a:局部代码块

在方法中出现；限定变量生命周期，及早释放，提高内存利用率

b:构造代码块 (初始化块)

在类中方法外出现；多个构造方法方法中相同的代码存放到一起，每次调用构造都执行，并且在构造方法前执行。

c:静态代码块

在类中方法外出现，并加上static修饰；用于给类进行初始化，在加载的时候就执行，并且只执行一次。

一般用于加载驱动

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_Code {

public static void main(String[] args) {

{int x = 10; //限定变量的声明周期

System.out.println(x); //输出语句不能写在外边，大括号里边的int x会被释放的，和for一样。}

Student s1 = new Student();

System.out.println("---------------");

Student s2 = new Student("张三",23);}

static {System.out.println("我是在主方法类中的静态代码块"); //优先于主方 法main执行的。

}

}

class Student {

private String name;

private int age;

public Student(){ //空参构造

//study();

System.out.println("空参构造");

}

public Student(String name,int age) { //有参构造

//study();

this.name = name;

this.age = age;

System.out.println("有参构造");

}

public void setName(String name) {

this.name = name; ..........

public int getAge() {

return age;

}

{ //构造代码块:每创建一次对象就会执行一次,优先于构造函数执行

//System.out.println("构造代码块");

study(); //构造代码块是构造方法中的共性，所以前边的都是可以省略的。

}

public void study() {

System.out.println("学生学习");

}

static { //随着类加载而加载,且只执行一次

System.out.println("我是静态代码块"); //作用:用来给类进行初始化,一般用来加载驱动

} //静态代码块是优先于主方法执行

}

---------------------------------------------------------------------------------------------------------------------------

### 8.02\_面向对象(代码块的面试题)

A:看程序写结果

---------------------------------------------------------------------------------------------------------------------------

class Student {

static {

System.out.println("Student 静态代码块");

}

{

System.out.println("Student 构造代码块");

}

public Student() {

System.out.println("Student 构造方法");

}

}

class Demo2\_Student {

static {

System.out.println("Demo2\_Student静态代码块");

}

public static void main(String[] args) {

System.out.println("我是main方法");

Student s1 = new Student();

Student s2 = new Student();

}

}

Demo2\_Student静态代码块

我是main方法

Student 静态代码块

Student 构造代码块

Student 构造方法

Student 构造代码块

Student 构造方法

---------------------------------------------------------------------------------------------------------------------------

## 继承

### //8.03\_面向对象(继承案例演示)

A:继承(extends)

让类与类之间产生关系,子父类关系

B:继承案例演示：

动物类,猫类,狗类

定义两个属性(颜色,腿的个数)两个功能(吃饭，睡觉)

C:案例演示

使用继承前

D:案例演示

使用继承后

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_Extends { 测试类！

public static void main(String[] args) {

Cat c = new Cat();

c.color = "花";

c.leg = 4;

c.eat();

c.sleep();

System.out.println(c.leg + "..." + c.color);

}

}

class Animal {

String color; //动物的颜色

int leg; //动物腿的个数

public void eat() { //吃饭的功能

System.out.println("吃饭");

}

public void sleep() { //睡觉的功能

System.out.println("睡觉");

}

}

class Cat extends Animal {

String color; //动物的颜色

int leg; //动物腿的个数

public void eat() { //吃饭的功能

System.out.println("吃饭");

}

public void sleep() { //睡觉的功能

System.out.println("睡觉");

}

}

class Dog extends Animal {

String color; //动物的颜色

int leg; //动物腿的个数

public void eat() { //吃饭的功能

System.out.println("吃饭");

}

public void sleep() { //睡觉的功能

System.out.println("睡觉");

}

}

extends是继承的意思

Animal是父类

Cat和Dog都是子类

---------------------------------------------------------------------------------------------------------------------------

### 8.04\_面向对象(继承的好处和弊端)

A:继承的好处

a:提高了代码的复用性

b:提高了代码的维护性，就是类与类之间的关系，同时也只一种弊端。

c:让类与类之间产生了关系，是多态的前提

B:继承的弊端

类的耦合性增强了。

**开发的原则：高内聚，低耦合。包装设计模式**

耦合：类与类的关系

内聚：就是自己完成某件事情的能力

### 8.05\_面向对象(Java中类的继承特点)

A:Java中类的继承特点

a:Java只支持单继承，不支持多继承。(一个儿子只能有一个亲爹)，多继承有安全的隐患的。

有些语言是支持多继承，格式：extends 类1,类2,...

b:Java支持多层继承(继承体系)，爷爷，父亲，自己，儿子，孙子等等。

c:继承只是一个相对的概念，父子也是相对的概念。你是他父亲也是别人的儿子。

d:多个类可以继承一个父类。一个父亲可以有很多的儿子。

B:Java中类的继承特点

如果想用这个体系的所有功能用最底层的类创建对象

如果想看这个体系的共性功能,看最顶层的类

### 8.06\_面向对象(继承的注意事项和什么时候使用继承)

A:继承的注意事项

a:子类只能继承父类所有非私有的成员(成员方法和成员变量)

b:子类不能继承父类的构造方法，但是可以通过super(马上讲)关键字去访问父类构造方法。（类名都不一样怎么去继承啊，通过super仅仅只是访问！并非继承）

c:不要为了部分功能而去继承（像底下的例子，父子类是一个所属的方法，is a关系。下边就只能是其他的人类什么的。）

项目经理 姓名 工号 工资 奖金

程序员 姓名 工号 工资

---------------------------------------------------------------------------------------------------------------------------

class Demo3\_Extends {

public static void main(String[] args) {

Son s = new Son(); //自己的构造方法。

s.show(); //调用继承父类而来的方法。

}

}

class Father {

**private String name; //私有成员变量，通过设置setget的公共访问方式让别人来访问，不能被继承。这里没有就setget方法就是不允许别人或者是子类来进行访问。**

**private void show() { //私有方法，下边再建立一个未被私有的方法来包含这个私有方法，通过访问非私有，达到访问私有的目的，绕了一圈。**

System.out.println("Hello World!");

}

}

class Son extends Father { //构造方法只是给自己本身来创建对象的使用的，不能继承的。再说了类名都不一样。

}

---------------------------------------------------------------------------------------------------------------------------

B:什么时候使用继承

继承其实体现的是一种关系："is a"。

Person

Student

Teacher

水果

苹果

香蕉

橘子

采用假设法。

如果有两个类A,B。只有他们符合A是B的一种，或者B是A的一种，就可以考虑使用继承。

### 8.07\_面向对象(继承中成员变量的关系)

A:案例演示

a:不同名的变量（先用子类自己的，子类没有的话采用父类的。）

b:同名的变量。子父类出现同名的变量只是在讲课中举例子有,在开发中是不会出现这种情况的，子类继承父类就是为了使用父类的成员,那么如果定义了同名的成员变量没有意义了。

### 8.08\_面向对象(this和super的区别和应用)

A:this和super都代表什么

this: 代表当前对象的引用,谁来调用我,我就代表谁

super: 代表当前对象父类的引用

B:this和super的使用区别

a:调用成员变量

**this.成员变量 调用本类的成员变量,也可以调用父类的成员变量**

**前提是本类没有！就是继承下来之后在调用的本类**

**super.成员变量 调用父类的成员变量，私有不能进行直接的调用**

---------------------------------------------------------------------------------------------------------------------------

class Father {

int num1 = 10;

int num2 = 30;

}

class Son extends Father {

int num2 = 20; //或者就是int s = super.num1.方法应该写在方法中

public void print() {

System.out.println(this.num1); //this既可以调用本类的,也可以调用父类的(本类没有的情况下)

System.out.println(this.num2); //就近原则,子类有就不用父类的了

System.out.println(super.num2); //调用的是父类的。

}

}

注意：类中只能是放类的成员或者是对象，不能是其他.

---------------------------------------------------------------------------------------------------------------------------

C:调用构造方法

this(...) 调用本类的构造方法

super(...) 调用父类的构造方法

D:调用成员方法

this.成员方法 调用本类的成员方法,也可以调用父类的方法

super.成员方法 调用父类的成员方法

### 8.09\_面向对象(继承中构造方法的关系)

A:案例演示

**子类中所有的构造方法**默认都会访问父类中**空参数的构造方法,若是没有空参构造的话需要写有参构造并且是去访问有参构造的**

B:为什么呢?

因为子类会继承父类中的数据，可能还会使用父类的数据。

所以，子类初始化之前，一定要先完成父类数据的初始化。

其实：

**每一个构造方法的第一条语句默认都是：super() Object类最顶层的父类。**

---------------------------------------------------------------------------------------------------------------------------

class Demo5\_Extends {

public static void main(String[] args) {

Son s = new Son();

}

}

class Father extends Object {

public Father() {

super();

System.out.println("Father 的构造方法");

}

}

class Son extends Father {

public Son() {

super(); //这是一条语句,如果不写,系统会默认加上,用来访问父类中的空参构造

System.out.println("Son 的构造方法");

}

}

---------------------------------------------------------------------------------------------------------------------------

### 8.10\_面向对象(继承中构造方法的注意事项)

A:父类没有无参构造方法,子类怎么办?

**父类没有无参构造方法,子类怎么办?（没有无参肯定有有参构造的，必有一个！）**

super解决：子类有参构造加super(name,age);而后通过get来获得。

this解决：就是通过this调用本类的有参构造来访问无参构造的。

总结：子类必须访问父类的构造方法！

B:this、super第一原则

super(…)或者this(….)必须出现在构造方法的第一条语句上。只能存在一个。

---------------------------------------------------------------------------------------------------------------------------

class Demo6\_Extends {

public static void main(String[] args) {

Son s1 = new Son(); //**记得默认的是子类的构造方法中都有父类的空参构造！也就说子类的有参构造也有父类的无参构造。**

System.out.println(s1.getName() + "..." + s1.getAge());

System.out.println("--------------------");

Son s2 = new Son("张三",23);

System.out.println(s2.getName() + "..." + s2.getAge());

}

}

class Father {

private String name; //姓名

private int age; //年龄

public Father() { //空参构造

System.out.println("Father 空参构造");

}

public Father(String name,int age) { //有参构造

this.name = name;

this.age = age;

System.out.println("Father 有参构造");

}

public void setName(String name) { //设置姓名

this.name = name; ........

public int getAge() { //获取年龄

return age;

}}

class Son extends Father {

public Son() { //空参构造

this("王五",25); //通过本类中的有参构造方法访问父类，绕了一圈。

//super("李四",24); //通过无参调用父类中的有参构造方法，这个也是可以的，但是只能传固定参数，不好。看下边。

System.out.println("Son 空参构造");

}

public Son(String name,int age) { //有参构造

super(name,age); //**父类中有有参的话，就通过这个方法解决。很简便，比上边的好。父类中私有了成员变量，所以这个地方不能直接的this。**

System.out.println("Son 有参构造");

}

}

---------------------------------------------------------------------------------------------------------------------------

### 8.11\_面向对象(继承中的面试题)

A:案例演示

看程序写结果1,this,super.

---------------------------------------------------------------------------------------------------------------------------

class Fu{

public int num = 10;

public Fu(){

System.out.println("fu");

}

}

class Zi extends Fu{

public int num = 20;

public Zi(){

System.out.println("zi");

}

public void show(){

int num = 30;

System.out.println(num);

System.out.println(this.num);

System.out.println(super.num);

}

}

class Test1\_Extends {

public static void main(String[] args) {

Zi z = new Zi();

z.show();

}

}

fu

Zi

30

20

10

---------------------------------------------------------------------------------------------------------------------------

看程序写结果2，代码块执行顺序

---------------------------------------------------------------------------------------------------------------------------

class Fu {

static {

System.out.println("静态代码块Fu");

}

{

System.out.println("构造代码块Fu");

}

public Fu() {

System.out.println("构造方法Fu");

}

}

class Zi extends Fu {

static {

System.out.println("静态代码块Zi");

}

{

System.out.println("构造代码块Zi");

}

public Zi() {

System.out.println("构造方法Zi");

}

}

Zi z = new Zi(); 请执行结果。

静态代码块Fu 静态代码块Zi 构造代码块Fu 构造方法Fu

构造代码块zi 构造方法Zi

1,jvm调用了main方法,main进栈

2,遇到Zi z = new Zi();会先将Fu.class和Zi.class分别加载进内存,再创建对象,当Fu.class加载进内存

父类的静态代码块会随着Fu.class一起加载,当Zi.class加载进内存,子类的静态代码块会随着Zi.class一起加载

第一个输出,静态代码块Fu,第二个输出静态代码块Zi

3,走Zi类的构造方法,因为java中是分层初始化的,先初始化父类,再初始化子类,所以先走的父类构造,但是在执行

父类构造时,发现父类有构造代码块,构造代码块是优先于构造方法执行的所以

第三个输出构造代码块Fu,第四个输出构造方法Fu

4,Fu类初始化结束,子类初始化,第五个输出的是构造代码块Zi,构造方法Zi

---------------------------------------------------------------------------------------------------------------------------

### 8.12\_面向对象(继承中成员方法关系)

A:案例演示

a:不同名的方法

b:同名的方法

---------------------------------------------------------------------------------------------------------------------------

class Demo7\_Extends {

public static void main(String[] args) {

Son s = new Son();

s.print();

s.method();

}

}

/\*

a:不同名的方法 **//谁有调谁。**

b:同名的方法 **//就是重写了父类的方法，肯定是调用自己的方法。但是还是想用父类的方法，就用super()；，下边就是调用案例**。

\*/

class Father {

public void print() {

System.out.println("Fu print");

}

}

class Son extends Father {

public void method() {

System.out.println("Zi Method");

}

public void print() {

**super.print(); //super可以调用父类的成员方法，若是父类私有就无法访问**

System.out.println("Zi print");

}

}

---------------------------------------------------------------------------------------------------------------------------

## 方法重写

### 8.13\_面向对象(方法重写概述及其应用)

A:什么是方法重写

重写:子父类出现了一模一样的方法(注意:返回值类型可以是子父类（就是子父类的实例对象，甚就是我们重写目前这个类的子父类对象）,这个我们学完面向对象讲)

B:方法重写的应用：

当子类需要父类的功能，而功能主体子类有自己特有内容时，可以重写父类中的方法。这样，即沿袭了父类的功能，又定义了子类特有的内容。Super访问原本的方法

C:案例演示

a:定义一个手机类。

---------------------------------------------------------------------------------------------------------------------------

class Demo7\_Phone {

public static void main(String[] args) {

Ios8 i = new Ios8();

i.siri();

i.call();

}

}

/\* ios7系统 siri speak English

ios8系统 siri 说中文 \*/

class Ios7 {

public void call() {

System.out.println("打电话");

}

public void siri() {

System.out.println("speak English");

}

}

class Ios8 extends Ios7 {

public void siri() {

System.out.println("说中文");

super.siri(); //可以放在后边，构造方法的时候放在第一行的。

}

}

---------------------------------------------------------------------------------------------------------------------------

### 8.14\_面向对象(方法重写的注意事项)

A:方法重写注意事项

a:父类中私有方法（若是想调用的话，就将此私有方法用this写到未被私有的方法中，然后在类的外部进行建立对象（静态了非私有方法就可以是类名（static）点调用的））不能被重写

因为父类私有方法子类根本就无法继承（通过上边的方式，若果是变量的话就是通过setget方法）

b:子类重写父类方法时，访问权限不能更低（父类是public，子类就不能是private），定义子类的目地就是比父类更牛逼，越来越小不可以的。

最好就一致

c:父类静态方法，子类也必须通过静态方法进行重写

其实这个算不上方法重写，但是现象确实如此，至于为什么算不上方法重写，多态中我会讲解(静态只能覆盖静态)（静态已经是存在得了，你建立对象来修改让别人怎么办）

子类重写父类方法的时候，最好声明一模一样。声明：参数类型、参数列表、返回值类型。

d:不能访问父类中被final修饰的。

B:案例演示

方法重写注意事项

---------------------------------------------------------------------------------------------------------------------------

class Demo8\_双桨 {

public static void main(String[] args) {

DayOne d = new DayOne();

d.泡妞();

d.print();

}

}

class 双桨 {

public void sing() {

System.out.println("唱红歌");

}

public void 泡妞() {

System.out.println("唱红歌搞定林夕合鸟女士");

}

public static void print() {

System.out.println("Fu print");

}

}

class DayOne extends 双桨 {

public void 泡妞() {

System.out.println("霸王硬上弓");

}

public static void print() { //静态只能覆盖静态,其实不算重写,多态时候详细讲解

System.out.println("Zi print");

}

}

---------------------------------------------------------------------------------------------------------------------------

### 8.15\_面向对象(方法重写的面试题)

A:方法重写的面试题

Override和Overload的区别?Overload能改变返回值类型吗?

overload可以改变返回值类型,只看参数列表

**方法重写**：子类中出现了和父类中方法声明一模一样的方法。与返回值类型有关,返回值是一致(或者是子父类)的。接口实现中的不是那么实现不是重写！

**方法重载**：本类中出现的方法名一样，参数列表不同（参数的类型、参数的列表、参数的顺序）的方法。与返回值类型无关。

**子类对象调用方法的时候：**

**先找子类本身，再找父类。**

### 8.16\_面向对象(使用继承前的学生和老师案例)

A:案例演示

使用继承前的学生和老师案例

属性:姓名,年龄

行为:吃饭

老师有特有的方法:讲课

学生有特有的方法:学习

---------------------------------------------------------------------------------------------------------------------------

class Student {

private String name; //姓名

private int age; //年龄

public Student() {} //空参构造

public Student(String name,int age) { //有参构造

this.name = name;

this.age = age;

}

public void setName(String name) { //设置姓名

this.name = name;

}

public int getAge() { //获取年龄

return age;

}

public void study() { //学习

System.out.println("学生学习");

}

}

---------------------------------------------------------------------------------------------------------------------------

### 8.17\_面向对象(使用继承后的学生和老师案例)

A:案例演示

使用继承后的学生和老师案例

---------------------------------------------------------------------------------------------------------------------------

class Person {

private String name; //姓名

private int age; //年龄

public Person() {} //空参构造

public Person(String name,int age) { //有参构造

this.name = name;

this.age = age;

}

public void setName(String name) { //设置姓名

this.name = name; ..........

public int getAge() { //获取年龄

return age;

}

public void eat() { //吃饭

System.out.println(name + "吃饭");

}

}

class Student extends Person {

public Student() {} //空参构造

public Student(String name,int age) {

super(name,age);

}

public void study() {

System.out.println(this.getName() + "学习");

}

}

class Teacher extends Person {

public Teacher() {} //空参构造

public Teacher(String name,int age) {

super(name,age);

}

public void teach() {

System.out.println(this.getName() + "讲课");

}

}

class Test4\_Person {

public static void main(String[] args) {

Student s1 = new Student();

s1.setName("张三");

s1.setAge(23);

System.out.println(s1.getName() + "..." + s1.getAge());

s1.eat();

s1.study();

System.out.println("------------------");

Student s2 = new Student("李四",24);

System.out.println(s2.getName() + "..." + s2.getAge());

s2.eat();

s2.study();

}

}

---------------------------------------------------------------------------------------------------------------------------

### 8.18\_面向对象(猫狗案例分析,实现及测试)

A:猫狗案例分析

B:案例演示

猫狗案例继承版

属性:毛的颜色,腿的个数

行为:吃饭

猫特有行为:抓老鼠catchMouse

狗特有行为:看家lookHome

---------------------------------------------------------------------------------------------------------------------------

class Test5\_Animal {

public static void main(String[] args) {

Cat c1 = new Cat("花",4);

System.out.println(c1.getColor() + "..." + c1.getLeg());

c1.eat();

c1.catchMouse();

Dog d1 = new Dog("黑",2);

System.out.println(d1.getColor() + "..." + d1.getLeg());

d1.eat();

d1.lookHome();

}

}

class Animal {

private String color; //毛的颜色

private int leg; //腿的个数

public Animal(){}

public Animal(String color,int leg) {

this.color = color;

this.leg = leg;

}

public void setColor(String color) { //设置颜色

this.color = color;

public int getLeg() { //获取腿的个数

return leg;

}

public void eat() { //吃饭

System.out.println("吃饭");

}

}

class Cat extends Animal {

public Cat() {} //空参构造

public Cat(String color,int leg) { //有参构造

super(color,leg);

}

public void eat() { //吃鱼

System.out.println("猫吃鱼");

}

public void catchMouse() { //抓老鼠

System.out.println("抓老鼠");

}

}

class Dog extends Animal {

public Dog() {} //空参构造

public Dog(String color,int leg) { //有参构造

super(color,leg);

}

public void eat() { //吃肉

System.out.println("狗吃肉");

}

public void lookHome() { //看家

System.out.println("看家");

}

}

---------------------------------------------------------------------------------------------------------------------------

## Final

### 8.19\_面向对象(final关键字修饰类,方法以及变量的特点)

A:final概述

B:final修饰特点

修饰类，类不能被继承。String类就是最终的类也是不能被继承的。

修饰变量，变量就变成了常量，只能被赋值一次，最初的成员变量默认值就成了无效值，final就只有一次机会你还给我一个无效值！这是不行的！

修饰方法，方法不能被重写，所有的方法都被final修饰那就将类final了。

C:案例演示

final修饰特点

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_Final {

public static void main(String[] args) {

Son s = new Son();

s.print();

}

}

/\*final class Father { **//最终的类**

public void print() {

System.out.println("访问底层数据资源");

}

}\*/

class Son /\*extends Father\*/ {

**final int NUM = 10;** //常量命名规范,如果是一个单词,所有字母大写,如果是多个单词,每个单词都大写,中间用下划线隔开。这就是自定义常量。

**public static final double PI = 3.14; //final修饰变量叫做常量,一般会与public static共用，方便调用，调用方法：类名.常量名**

public void print() {

//NUM = 20;

System.out.println(NUM);

}

}

---------------------------------------------------------------------------------------------------------------------------

### 8.20\_面向对象(final关键字修饰局部变量)

A:案例演示

方法内部或者方法声明上都演示一下(了解)

**基本类型，是值不能被改变**

**引用类型，是地址值不能被改变,对象中的属性可以改变**

---------------------------------------------------------------------------------------------------------------------------

class Demo2\_Final {

public static void main(String[] args) {

**final int num = 10; //一定要初始化，不能是默认的！**

**//num = 20; //修改常量呢？坚决不行！**

**System.out.println(num);**

**final Person p = new Person("张三",23);**

**//p = new Person("李四",24); //这是在修改地址呢！**

p.setName("李四"); //这才是修改属性值

p.setAge(24);

System.out.println(p.getName() + "..." + p.getAge());

method(10);

method(20); //上边的弹栈了，重来一次必须可以啊！

}

public static void method(final int x) {

System.out.println(x);

}

}

class Person {

private String name; //姓名

private int age; //年龄

public Person(){} //空参构造

public Person(String name,int age) {

this.name = name;

this.age = age;

}

public void setName(String name) { //设置姓名

this.name = name;

public int getAge() { //获取年龄

return age;

}

}

---------------------------------------------------------------------------------------------------------------------------

### 8.21\_面向对象(final修饰变量的初始化时机)

A:final修饰变量的初始化时机

显示初始化

在对象构造完毕前即可

---------------------------------------------------------------------------------------------------------------------------

class Demo3\_Final {

public static void main(String[] args) {

Demo d = new Demo();

d.print();

}

}

/\*

A:final修饰变量的初始化时机

显示初始化

在对象构造完毕前即可 //两者不能同事进行的

\*/

class Demo {

final int num = 10; //成员变量的默认初始化值是无效值

public Demo() {

num = 10; //在对象构造完毕前即可

}

public void print() {

System.out.println(num);

}

}

---------------------------------------------------------------------------------------------------------------------------

DAY09 多态中的成员访问特点、静态方法，自动与强转。抽象类中的特点。接口中特点。

## 多态

### 9.01\_面向对象(多态的概述及其代码体现)

A:多态(polymorphic)概述

\* 对象在不同的时刻表现出来的不同的状态。

B:多态前提

\* a:要有继承关系。

\* b:要有方法重写。**(若是没有重写的话就没有意义了，就是为了重写的好方法的)**

**\* c:要有父类引用指向子类对象。**

C:案例演示

\* 代码体现多态

class Demo1\_Polymorphic {

public static void main(String[] args) {

Cat c = new Cat();

c.eat();

**Animal a = new Cat(); //父类引用指向子类对象**

a.eat();

}

}

class Animal {

public void eat() {

System.out.println("动物吃饭");

}

}

class Cat extends Animal {

public void eat() {

System.out.println("猫吃鱼");

}

}

### 9.02\_面向对象(多态中的成员访问特点之成员变量)

成员变量

\* 编译看左边(父类)，运行看左边(父类)。多态中成员变量图片。

### 9.03\_面向对象(多态中的成员访问特点之成员方法)

成员方法

**\* 编译看左边(父类)，运行看右边(子类)（子类没有继续运行父类。子类继承了父类非私有的（或者未被其他修饰的）方法）。动态绑定。**多态中成员方法图片。

### 9.04\_面向对象(多态中的成员访问特点之静态成员方法)

静态方法

\* 编译看左边(父类)，运行看左边(父类)。

**\* (静态和类相关，随类加载的不是重写，是覆盖。所以，访问还是左边的)**

\* 只有非静态的成员方法,编译看左边,运行看右边

class Demo2\_Polymorphic {

public static void main(String[] args) {

Father f = new Son(); //父类引用指向子类对象，f是父类对象看super

System.out.println(f.num); //输出为10

Son s = new Son(); //s是子类对象，可以看到全局

System.out.println(s.num)； //输出为20.这两种注意看内存图画法。

Father f = new Son();

f.print(); //只有非静态的成员方法,编译看左边,运行看右边son

f.method(); //相当于是Father.method(),就是静态成员还是自己father static method。}}

class Father {

int num = 10;

public void print() {

System.out.println("father");

}

public static void method() {

System.out.println("father static method");

}}

class Son extends Father {

int num = 20;

public void print() {

System.out.println("son");}

public static void method() {

System.out.println("son static method");

}}

### 9.05\_面向对象(超人的故事)

A:案例分析

class Demo3\_SuperMan {

public static void main(String[] args) {

**Person p = new SuperMan(); //父类引用指向子类对象,超人提升为了人**

**//父类引用指向子类对象就是向上转型**

**System.out.println(p.name);**

**p.谈生意();**

**SuperMan sm = (SuperMan)p; //向下转型**

**sm.fly();**

/\*

基本数据类型自动类型提升和强制类型转换

\*/

int i = 10;

byte b = 20;

//i = b; //自动类型提升

//b = (byte)i; //强制类型转换

}

}

class Person {

String name = "John";

public void 谈生意() {

System.out.println("谈生意");

}

}

class SuperMan extends Person {

String name = "superMan";

public void 谈生意() {

System.out.println("谈几个亿的大单子");

}

public void fly() {

System.out.println("飞出去救人");

}

}

### 9.06\_面向对象(多态中向上转型和向下转型)

A:案例演示

\* 详细讲解多态中向上转型和向下转型

**Person p = new SuperMan();向上转型**

**SuperMan sm = (SuperMan)p;向下转型**

### 9.07\_面向对象(多态的好处和弊端)

A:多态的好处

**\* a:提高了代码的维护性(继承保证)（父类改了子类也跟着该这就是继承保证的意思）**

**\* b:提高了代码的扩展性(由多态保证)（举例证明，下边）**

B:案例演示

\* 多态的好处

\* 可以当作形式参数,可以接收任意子类对象 method(Animal a)形式参数。

C:多态的弊端

**\* 不能使用子类的特有属性和行为。（注意此处强调的是特有的！若是重写父类就还是可以调用的，但是编译用的是父类，运行的子类，动态绑定。属性也是一样的，不能访问子类自己增加属性），若是要使用子类的私有方法或者成员必须通过强转才可以调用。**

D:案例演示

method(Animal a)

method(Cat c)

class Demo4\_Animal {

public static void main(String[] args) {

method(new Cat());

method(new Dog());

method(new Pig());

/\*public static void method(Cat c) {

c.eat();

}

public static void method(Dog d) {

d.eat();}\*/

**public static void method(Animal a) //当作参数的时候用多态最好,因为扩展性强**

a.eat(); 等等重写的方法。 非常方便

当做参数的话中间的method方法就不用重复的写，而且再增加子类的时候还要写就麻烦了，就这这样用父类传参非常方便，提供了入口。再用到前边来创建并通过method调用，就可运行子类的方法（通过改写父类的方法而来的方法，私有的还是不行）

class Demo4\_Animal {

public static void main(String[] args) {

//Cat c1 = new Cat();

//c1.eat();

**//Animal a = new Cat(); 开发的是很少在创建对象的时候用父类引用指向子类对象,直接创建子类对象更方便,可以使用子类中的特有属性和行为**

}

/\*public static void method(Cat c) {

c.eat();

}

public static void method(Dog d) {

d.eat();}\*/

**public static void method(Animal a) {** //当作参数的时候用多态最好,因为扩展性强

**//关键字 instanceof 判断前边的引用是否是后边的数据类型**

**if (a instanceof Cat) { //强转型！**

**Cat c = (Cat)a;**

**c.eat();**

**c.catchMouse();**

**}else if (a instanceof Dog) {**

**Dog d = (Dog)a;**

**d.eat();**

**d.lookHome();**

**}else {**

**a.eat();}}}**

class Animal {

public void eat() {

System.out.println("动物吃饭");}}

class Cat extends Animal {

public void eat() {

System.out.println("猫吃鱼");}

public void catchMouse() {

System.out.println("抓老鼠");}}

class Dog extends Animal {

public void eat() {

System.out.println("狗吃肉");

}

public void lookHome() {

System.out.println("看家");

}

}

如果只写一类的话，就会把狗强转成猫就会出现类型转换异常,ClassCastException。所以在这里才会出现这样的选择式写法。

强转才可以调用了私有方法！

### 9.08\_面向对象(多态中的题目分析题)

A:看下面程序是否有问题，如果没有，说出结果

class Fu {

public void show() {

System.out.println("fu show");

}

}

class Zi extends Fu {

public void show() {

System.out.println("zi show");

}

public void method() {

System.out.println("zi method");

}

}

class Test1Demo {

public static void main(String[] args) {

Fu f = new Zi();

f.method(); //编译报错

f.show(); //zi show，子类么有重写直接继承父类也行

}

}

B:看下面程序是否有问题，如果没有，说出结果

class A {

public void show() {

show2();

}

public void show2() {

System.out.println("我");

}

}

class B extends A {

public void show2() {

System.out.println("爱");

}

}

class C extends B {

public void show() {

super.show();

}

public void show2() {

System.out.println("你");

}

}

public class Test2DuoTai {

public static void main(String[] args) {

A a = new B();

a.show(); //爱

B b = new C();

b.show(); //你，**父类的show是调用子类的show2**  } //**只是取了一个方法。父类的编译也是必须的通过才行**

}

Show被继承，show2被重写覆盖。

## 抽象类

### 9.09\_面向对象(抽象类的概述及其特点)

A:抽象类概述

\* 抽象就是看不懂的

B:抽象类特点

\* a:抽象类和抽象方法必须用abstract关键字修饰

**\* abstract class 类名 {} 抽象类**

**\* public abstract void eat(); 抽象方法**

\* b:**抽象类不一定有抽象方法（可以一个抽象方法都没有），有抽象方法的类一定是抽象类或者是接口（接口当中全部是抽象类）。**

\* c:抽象类不能实例化那么，抽象类如何实例化呢?

不能实例化，若是可以的话就可以调用方法，但是方法根本就是不能用的。

\* **按照多态的方式，由具体的子类实例化。其实这也是多态的一种，抽象类多态。**

\* d:抽象类的子类

\* 要么是抽象类（还是不能进行实例化，只是继承了abstract而已）

**\*要么重写抽象类中的所有抽象方法（成为普通类，必须重写所有的方法！）常用**

C:案例演示

\* 抽象类特点

### 09.10\_面向对象(抽象类的成员特点)

A:抽象类的成员特点

**\* a:成员变量：既可以是变量，也可以是常量。abstract是否可以修饰成员变量?不能修饰成员变量，能修饰成员方法。**

**\* b:构造方法：有。接口中没有构造方法的。**

**\* 用于子类访问父类数据的初始化。**

**\* c:成员方法：既可以是抽象的，也可以是非抽象的。**

B:案例演示

\* 抽象类的成员特点

C:抽象类的成员方法特性：

**\* a:抽象方法 强制要求子类做的事情。子类同时也能做父类以外的其他事情。**

**\* b:非抽象方法 子类继承的事情，提高代码复用性。**

abstract class Demo {

int num1 = 10;

final int num2 = 20; //具体的数值怎么去抽象，所以不能抽象成员变量

public Demo(){}

public void print() { //非私有默认的继承

System.out.println("111");

}

public abstract void method();

}

class Test extends Demo {

public void method() { //子类不是抽象类那么就要重写所有的抽象方法。

System.out.println("111");

}

}

### ////09.11\_面向对象(葵花宝典)

案例演示

\* 抽象类的作用

class Demo3\_葵花宝典 {

public static void main(String[] args) {

岳不群 小岳子 = new 岳不群();

小岳子.自宫();

}

}

abstract class 葵花宝典 {

public abstract void 自宫(); //不知道后边是怎么操作的。

}

class 岳不群 extends 葵花宝典 {

public void 自宫() {

System.out.println("用牙签");

}

}

class 林平之 extends 葵花宝典 {

public void 自宫() {

System.out.println("用指甲刀");

}

}

class 东方不败 extends 葵花宝典 {

public void 自宫() {

System.out.println("用锤子,不忍直视");

}

}

### ////09.12\_面向对象(抽象类练习猫狗案例)

A:案例演示

\* 具体事物：猫，狗

\* 共性：姓名，年龄，吃饭

\* 猫的特性:抓老鼠

\* 狗的特性:看家

class Test1\_Animal {

public static void main(String[] args) {

Cat c = new Cat("加菲",8);

System.out.println(c.getName() + "..." + c.getAge());

c.eat();

c.catchMouse();

Dog d = new Dog("八公",30); //注意这里并不是多态的。

System.out.println(d.getName() + "..." + d.getAge());

d.eat();

d.lookHome();

}

}

abstract class Animal {

private String name; //姓名

private int age; //年龄

public Animal(){} //空参

public Animal(String name,int age) {//有参

this.name = name;

this.age = age;

}

public void setName(String name) { //设置姓名

this.name = name;

public int getAge() { //获取年龄

return age;

}

public abstract void eat(); //吃饭

}

class Cat extends Animal {

public Cat(){} //空参

**public Cat(String name,int age) { //有参**

**super(name,age);**

**}**

public void eat() {

System.out.println("猫吃鱼");

}

public void catchMouse() {

System.out.println("抓老鼠");

}

}

class Dog extends Animal {

public Dog(){} //空参

public Dog(String name,int age) {//有参

super(name,age);

}

public void eat() {

System.out.println("狗吃肉");

}

public void lookHome() {

System.out.println("看家");

}

}

还有私人参数，传进去，但是不写进super，写在后边外边。

### ////09.13\_面向对象(抽象类练习老师案例)

A:案例演示

\* 具体事物：基础班老师，就业班老师

\* 共性：姓名，年龄，讲课。

\* 具体事物:基础班学生,就业班学生

\* 共性:姓名,年龄,学习

class Test2\_Teacher {

public static void main(String[] args) {

BaseTeacher bt = new BaseTeacher("冯佳",18);

bt.teach();

}

}

abstract class Teacher {

private String name; //姓名

private int age; //年龄

public Teacher(){} //空参

public Teacher(String name,int age) {//有参

this.name = name;

this.age = age;

}

public void setName(String name) { //设置姓名

this.name = name;

public int getAge() { //获取年龄

return age;

}

public abstract void teach();

}

class BaseTeacher extends Teacher {

**public BaseTeacher(){} //空参**

**public BaseTeacher(String name,int age) {//有参**

**super(name,age);**

**}**

public void teach() {

System.out.println("我的姓名是:" + **this.getName()** + ",我的年龄是:" + this.getAge() + ",讲的内容是java基础");

}

}

### ////09.14\_面向对象(抽象类练习员工案例)

A:案例演示

\* 假如我们在开发一个系统时需要对程序员类进行设计，程序员包含3个属性：姓名、工号以及工资。

\* 经理，除了含有程序员的属性外，另为还有一个奖金属性。

\* 请使用继承的思想设计出程序员类和经理类。要求类中提供必要的方法进行属性访问。

class Test3\_Employee {

public static void main(String[] args) {

Coder c = new Coder("德玛西亚","007",8000);

c.work();

Manager m = new Manager("苍老师","9527",3000,20000);

m.work();

}

}

abstract class Employee {

private String name; //姓名

private String id; //工号

private double salary; //工资

public Employee() {} //空参构造

public Employee(String name,String id,double salary) {

this.name = name;

this.id = id;

this.salary = salary;

}

public void setName(String name) { //设置姓名

this.name = name;

public double getSalary() { //获取工资

return salary;

}

public abstract void work();

}

//程序员

class Coder extends Employee {

public Coder() {} //空参构造

public Coder(String name,String id,double salary) {

super(name,id,salary);

}

public void work() {

System.out.println("我的姓名是:" + this.getName() + ",我的工号是:" + this.getId() + ",我的工资是:"

+ this.getSalary() + ",我的工作内容是敲代码");

}

}

//项目经理

class Manager extends Employee {

private int bonus; //奖金

public Manager() {} //空参构造

public Manager(String name,String id,double salary,int bonus) {

super(name,id,salary);

this.bonus = bonus;

}

public void work() {

System.out.println("我的姓名是:" + this.getName() + ",我的工号是:" + this.getId() + ",我的工资是:"

+ this.getSalary() + ",我的奖金是:" + bonus + ",我的工作内容是管理");

}

}

### 09.15\_面向对象(抽象类中的面试题)

A:面试题1

\* 一个抽象类如果没有抽象方法，可不可以定义为抽象类?如果可以，有什么意义?

\* 可以

\* 这么做目的只有一个,就是不让其他类创建本类对象,交给子类完成

B:面试题2不共存，static，final，private。

\* abstract不能和哪些关键字共存

**abstract和static**

被abstract修饰的方法没有方法体

被static修饰的可以用类名.调用,但是类名.调用抽象方法是没有意义的，没有方法体。

**abstract和final**

被abstract修饰的方法强制子类重写

被final修饰的不让子类重写,所以他俩是矛盾

**abstract和private**

被abstract修饰的是为了让子类看到并强制重写

被private修饰不让子类访问,所以他俩是矛盾的

## 接口

### 09.16\_面向对象(接口的概述及其特点)

A:接口概述

\* 从狭义的角度讲就是指java中的interface，interface就是abstract的abstract，抽象的抽象。

\* 从广义的角度讲对外提供规则的都是接口

B:接口特点

\* a:接口用关键字interface表示

**\* interface 接口名 {} （和类名差不多）**

\* b:类实现接口用implements表示

**\* class 类名 implements 接口名 {}**

\* c:接口不能实例化，方法不能用和abstract一样。

\* 那么，接口如何实例化呢?

\* 按照多态的方式来实例化。

**\* d:接口的子类**

\* a:可以是抽象类。但是意义不大。

**\* b:可以是具体类。要重写接口中的所有抽象方法。(推荐方案)**

C:案例演示

\* 接口特点

class Demo1\_Interface {

public static void main(String[] args) {

**//Inter i = new Inter(); //接口不能被实例化,因为调用抽象方法没有意义**

**Inter i = new Demo(); //父类引用指向子类对象，和继承一样。**

i.print();

}

}

interface Inter {

public abstract void print(); //接口中的方法都是抽象的

}

class Demo implements Inter { **//这里的implement就相当于是extends，继承类的父子关系一样。implement来实例化接口，就同子父类一样，只是另外的一种形式的父子。**

public void print() {

System.out.println("print");

}

}

**普通类之间，抽象普通类之间是继承extends.而接口和普通类是实现implements，两者是可以共存的**

### 09.17\_面向对象(接口的成员特点)

A:接口成员特点。对外原则是暴露的，

**\* 成员变量；只能是常量，并且是静态的并公共的。**

**\* 默认修饰符：public static final，位置随便换**

**\* 建议：自己手动给出。**

**\* 构造方法：接口没有构造方法。抽象类中有的**

**\* 成员方法：只能是抽象方法，不能有方法体{}。**

**\* 默认修饰符：public abstract**

**\* 建议：自己手动给出。**

B:案例演示

\* 接口成员特点

class Demo2\_Interface {

public static void main(String[] args) {

Demo d = new Demo();

d.print();

System.out.println(Inter.num);

}

}

interface Inter {

**public static final int num = 10; //接口是暴露的，所有的成员前边的修饰符都是public。**

//public Inter(){} **接口中没有构造方法，儿子有亲爹object。**

/\*public void print() { 接口中不能定义非抽象方法

}\*/

public abstract void print();

}

class Demo /\*extends Object\*/ implements Inter { **//一个类不写继承任何类,默认继承Object类。有亲爹不妨碍他认干爹！多个干爹也是可以的。**

public void print() { //父类中的public是默认的，但是子类中并没有默认，所以父类没有你子类还是得写上权限，不然默认的default是小于public的。

//num = 20; 不能二次赋值，第一次之后已经是常量了。

System.out.println(num);

}

public Demo() {

super();

}

}

### 09.18\_面向对象(类与类,类与接口,接口与接口的关系)

A:类与类,类与接口,接口与接口的关系

\* a:类与类：

\* 继承关系,只能单继承,可以多层继承。

\* b:类与接口：

\* 实现关系,可以单实现,也可以多实现。没有方法体必须可以。

\* 并且还可以在继承一个类的同时实现多个接口。这就打破了类之间的单继承的缺点。

\* c:接口与接口：

\* 继承关系,可以单继承,也可以多继承。

B:案例演示

\* 类与类,类与接口,接口与接口的关系

interface InterA {

public abstract void printA();

}

interface InterB {

public abstract void printB();

}

interface InterC extends InterB,InterA {

}

//class Demo implements InterA,implements InterB { //这么做不允许是非法的

**class Demo extends Object implements InterA,InterB** {

public void printA() {

System.out.println("printA");

}

public void printB() {

System.out.println("printB");

}

}

必须对接口进行方法的重写。

### 09.19\_面向对象(抽象类和接口的区别)

A:成员区别

\* 抽象类：

\* 成员变量：可以变量，也可以常量

\* 构造方法：有

\* 成员方法：可以抽象，也可以非抽象

\* 接口：

\* 成员变量：只可以常量

\* 成员方法：只可以抽象

B:关系区别

\* 类与类

\* 继承，单继承

\* 类与接口

\* 实现，单实现，多实现

\* 接口与接口

\* 继承，单继承，多继承

C:设计理念区别

**\* 抽象类 被继承体现的是：”is a”的关系。抽象类中定义的是该继承体系的共性功能。**

**\* 接口 被实现体现的是：”like a”的关系。接口中定义的是该继承体系的扩展功能。**

### 09.20\_面向对象(猫狗案例加入跳高功能分析及其代码实现)

A:案例演示

\* 动物类：姓名，年龄，吃饭，睡觉。

\* 猫和狗

\* 动物培训接口：跳高

class Test1\_Animal {

public static void main(String[] args) {

Animal c = new Cat("加菲",8);

c.eat();

c.sleep();

JumpCat jc = new JumpCat("跳高猫",3);

jc.eat();

jc.sleep();

jc.jump();

}

}

abstract class Animal {

private String name; //姓名

private int age; //年龄

public Animal() {} //空参构造

public Animal(String name,int age) {//有参构造

this.name = name;

this.age = age;

}

public void setName(String name) { //设置姓名

this.name = name;

public int getAge() { //获取年龄

return age;

}

public abstract void eat(); //吃饭

public abstract void sleep(); //睡觉

}

interface Jumping { //跳高的接口

public void jump();

}

class Cat extends Animal {

public Cat() {} //空参构造

public Cat(String name,int age) {//有参构造

super(name,age);

}

public void eat() {

System.out.println("猫吃鱼");

}

public void sleep() {

System.out.println("侧着睡");

}

}

class JumpCat extends Cat implements Jumping {

public JumpCat() {} //空参构造

public JumpCat(String name,int age) {//有参构造

super(name,age);

}

public void jump() {

System.out.println("猫跳高");

}

}

DAY10 包操作，权限修饰符，局部内部类及匿名内部类，成员内部类。

包操作及详细讲解权限修饰符

## 包简单知识

### ////10.01\_面向对象(package关键字的概述及作用)

A:为什么要有包

将字节码(.class)进行分类存放

包其实就是文件夹

包格式：com.heima.add(teacher)。就是域名的反转，后边要么是按照功能要么是按照模块。包名是全部小写。

B:包的概述

举例：

学生：增加，删除，修改，查询

老师：增加，删除，修改，查询

...

方案1：按照功能分

com.heima.add

AddStudent

AddTeacher

方案2：按照模块分

com.heima.teacher

AddTeacher

DeleteTeacher

UpdateTeacher

FindTeacher

### ////10.02\_面向对象(包的定义及注意事项)

A:定义包的格式

package 包名;

多级包用.分开即可（.就是路径分隔符，一般肯定是多级路径，条理清楚啊）

B:定义包的注意事项

A:package语句必须是程序的第一条可执行的代码

B:package语句在一个java文件中只能有一个

C:如果没有package，默认表示无包名

D:在实际的开发中其实是一个原文件中只放一个类放，便于管理。

**全类名：就是包名加类名。（是一种比较全的类名）**

### ////10.03\_面向对象(带包的类编译和运行)

A:如何编译运行带包的类

a:javac编译的时候带上-d即可

javac -d . HelloWorld.java（若是直接运行还是可以的只是生成的.class文件还是在当前的文件夹下，那么你写包就没有什么用途了。按照格式运行就会在你设置好的文件夹下import com.baidu，这个包名就是在当前（若d后边是点，也可以是e/指的就是e盘了）文件夹下的com文件夹下baidu文件夹下文件。）

Eg:javac -d . com.hei.HelloWorld.java(原Java文件包名就是com.hei 类名就是Helloworld)。运行就在本目录下不进包里边去。

b:通过java命令执行。

java 包名.HellWord（全包名）

### ////10.04\_面向对象(不同包下类之间的访问)

A:案例演示

不同包下类之间的访问

注意：1、若是基本类文件不编译直接放在和测试类一个文件夹下，那么在编译测试类文件的时候直接就可以将基本类文件直接的编译。运行是没有问题的。若是在不同的文件下是不行的。2、不同包之间不能直接编译应该加上层级目录，就是导包。

### ////10.05\_面向对象(import关键字的概述和使用)

A:案例演示

为什么要有import

其实就是让有包的类对调用者可见,不用写全类名了，就是包就行了。

B:导包格式

import 包名;

注意：

这种方式导入是到类的名称。

虽然可以最后写\*，但是不建议。

---------------------------------------------------------------------------------------------------------------------------

package com.xxx;

import com.baidu.Person;导包就是倒入具体的类，不建议下边的写法，不快！

import com.baidu.\*;

--------------------------------------------------------------------------------------------------------------------------

C:package,import,class有没有顺序关系(面试题)

## 权限修饰符

### 10.06\_面向对象(四种权限修饰符的测试)

A:案例演示

四种权限修饰符

B:结论

**本类 同一个包下(子类和无关类) 不同包下(子类) 不同包下(无关类)**

**private Y**

**默认 Y Y**

**protected Y Y Y**

**public Y Y Y Y**

**不同包下的无关类指的是直接的创建对象，没有父子关系的。**

**解释：**

**private最吝啬就是我只是给我自己用的，除了我这个类谁都别想，不管你是不是我自己的子类，谁继承访问都不行！除非我将自己的变量和方法提供给我自己公共方法当然其中我是写了限制条件的，你访问必须的满足我公共方法的条件，还是比较吝啬。**

**默认就是什么修饰也不加，我自己和我自己的子类，还有就是住在一个包下边的，你们都可以访问我。**

**Protected，比默认大方点，我的子类跑到别的包下我也让你用。保护部分成员的。**

**Public：最大方，你们谁都可以用，真正的共享！**

### 10.07\_面向对象(类及其组成所使用的常见修饰符)

A:修饰符：

权限修饰符：private，默认的，protected，public

状态修饰符：static，final（安静的躺着或者是最终的放在那里）

抽象修饰符：abstract

B:类：

权限修饰符：默认修饰符，public（私有类就是没有创建类，Protected保护部分成员的）

状态修饰符：final （方法final能继承不能重写，都不能重写那你就别被继承了）

抽象修饰符：abstract （只能是抽象方法和抽此类）

用的最多的就是：public

C:成员变量：

权限修饰符：private，默认的，protected，public

状态修饰符：static，final

用的最多的就是：private

D:构造方法：

权限修饰符：private，默认的，protected，public

用的最多的就是：public

E:成员方法：

权限修饰符：private，默认的，protected，public

状态修饰符：static，final

抽象修饰符：abstract

用的最多的就是：public

F:除此以外的组合规则：

成员变量：public static final （举例final static final int MAX\_VALUE = 5;）

成员方法：

public static

public abstract

public final

## 内部类:

### 10.08\_面向对象(内部类概述和访问特点)

A:内部类概述

在类的内部再定义类，里面的类就是内部类。

B:内部类访问特点

a:内部类可以直接访问外部类的成员，包括私有。

b:外部类要访问内部类的成员，必须创建对象。

外部类名.内部类名 对象名 = 外部类对象.内部类对象;

对象的创建一般都是在测试类中的，你在基本类中创建，也是可以的，但是你如果有方法的执行，就是错误的，因为jvm只认识主函数啊，执行不到你，你就是错误的。那么久继续在测试类创建对象再调用，直接在测试类中创建不就一次到位了。

C:案例演示

内部类极其访问特点

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_InnerClass {

public static void main(String[] args) {

/\*

Inner i = new Inner();

i.method(); 直接创建应该是外部类的特性，内部不能这样子。

\*/

//外部类名.内部类名 = 外部类对象.内部类对象

**Outer.Inner oi = new Outer().new Inner(); //创建内部类对象**

//创建外部类new Outer()，发现里边还有类，就再创建new Inner。完事这就是内部类。

//点，在这里是一种层级关系。

oi.method();

}

}

class Outer {

private int num = 10;

class Inner { **//这是成员内部类，将这个类看作是成员方法即可6。**

public void method() {

System.out.println(num);

}

}

}

--------------------------------------------------------------------------------------------------------------------------

### 10.09\_面向对象(成员内部类私有使用)

Private，本类中创建未被私有的方法包含了私有的类的对象

---------------------------------------------------------------------------------------------------------------------------

class Demo2\_InnerClass {

public static void main(String[] args) {

/\*

Outer.Inner oi = new Outer().new Inner();

oi.method(); //私有内部肯定不能直接创建！

\*/

Outer o = new Outer();

o.print();

}

}

class Outer {

private int num = 10;

private class Inner {

public void method() {

System.out.println(num);

}

}

//因为类这在里是属于成员变量的，所以方法可以直接访问。

public void print() { **//通过在本类中建立非私有的方法来建立私有类的对象。然后通过访问非私有的方法进行建立私有的对象，然后就可以访问私有类里边的方法及变量。**

Inner i = new Inner();

i.method();

}

}

--------------------------------------------------------------------------------------------------------------------------

### 10.10\_面向对象(静态成员内部类)

static

B:成员内部类被静态修饰后的访问方式是:

外部类名.内部类名 对象名 = 外部类名.内部类对象;

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_InnerClass {

public static void main(String[] args) {

**//外部类名.内部类名 对象名 = 外部类名.内部类对象;**

**Outer.Inner oi = new Outer.Inner();//这里的static指的是内部类所以没有new，直接进行访问。后边的new其实就是inner的，并不是外部的。**

**oi.method();**

**Outer.Inner2.print();**

}

}

class Outer {

static class Inner { //静态给的只是外部并不是内部。

public void method() { //非静方法

System.out.println("method");

}

}

static class Inner2 { //都是静态一路狂奔到最里边。

public static void print() { //静态的方法

System.out.println("print");

}

}

}

--------------------------------------------------------------------------------------------------------------------------

### 10.11\_面向对象(成员内部类的面试题)

A:面试题

要求：使用已知的变量，在控制台输出30，20，10。

**内部类之所以能获取到外部类的成员,是因为他能获取到外部类的引用外部类名.this**

class Outer {

public int num = 10;

class Inner {

public int num = 20;

public void show() {

int num = 30;

System.out.println(?); //num

System.out.println(??); //this.num

System.out.println(???); //outer.this.num（并非继承）

}

}

}

class InnerClassTest {

public static void main(String[] args) {

Outer.Inner oi = new Outer().new Inner();

oi.show();

}

}

### 10.12\_面向对象(局部内部类访问局部变量的问题)

A:案例演示

局部内部类访问局部变量必须用final修饰

局部内部类在访问他所在方法中的局部变量必须用final修饰,为什么?

因为当调用这个方法时,局部变量如果没有用final修饰,他的生命周期和方法的生命周期是一样的,当方法弹栈,这个局部变量也会消失,那么如果局部内部类对象还没有马上消失，还想用这个局部变量,就没有了,如果用final修饰会在类加载的时候进入常量池,即使方法弹栈,常量池的常量还在,也可以继续使用

但是jdk1.8取消了这个事情,所以我认为这是个bug

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_InnerClass {

public static void main(String[] args) {

Outer o = new Outer();

o.method();

}

}

//局部内部类，就是相当用于局部的变量。

class Outer {

public void method() {

final int num = 10;

class Inner {

public void print() {

System.out.println(num);

}

}

Inner i = new Inner(); //这才是局部内部类

i.print(); //创建之后又进行自我的访问。

}

/\*public void run() {

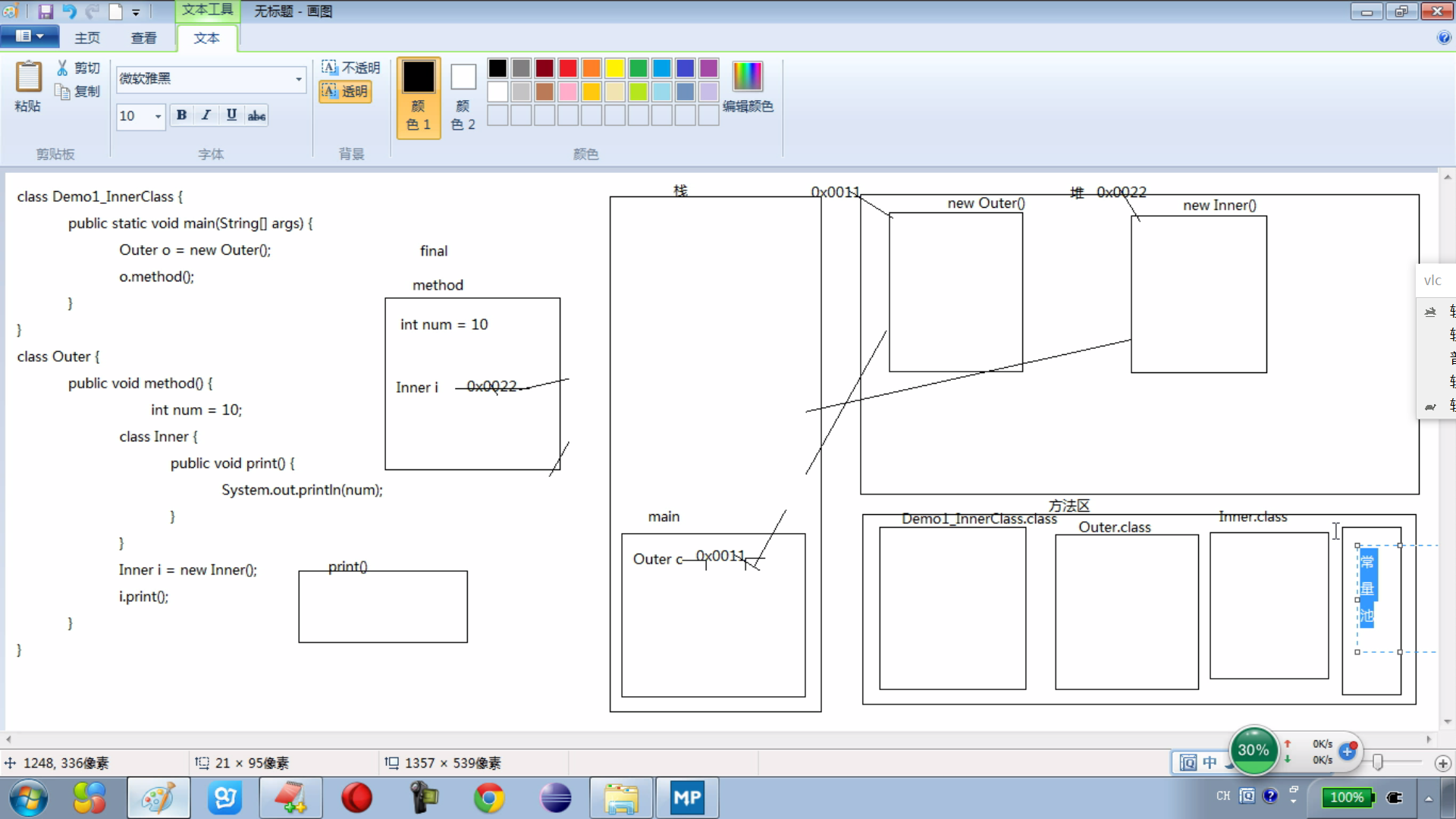
Inner i = new Inner(); //局部内部类,只能在其所在的方法中访问

i.print();

}\*/

}

--------------------------------------------------------------------------------------------------------------------------



### 10.13\_面向对象(匿名内部类的格式和理解)

A:匿名内部类（局部内部类的一种）

就是内部类的简化写法。

B:前提：存在一个类或者接口

这里的类可以是具体类也可以是抽象类。

C:格式：

new 普通类名/抽象类名/接口名() {

//重写的方法;

一些内容;

};

D:本质是什么呢?

是一个继承了该类或者实现了该接口的子类匿名对象,为了书写方便就是不再单独的定义一个类(这个类实现了接口或者继承了抽象类或者普通类)又去创建对象再然后创建对象来调用重写的方法,直接固定成一种简便的格式!***更应该称作匿名子类对象***.特点,就是只能是调用一次,只是在测试的时候用到,而且还是当前提条件的类或者是抽象类或者是接口中只有一个方法的时候用到.下边只是一种推导的过程,可以不用管.

E:案例演示

按照要求来一个匿名内部类

---------------------------------------------------------------------------------------------------------------------------

class Demo1\_NoNameInnerClass {

public static void main(String[] args) {

Outer o = new Outer();

o.method();

}

}

interface Inter {

public void print();

}

class Outer {

class Inner implements Inter { //类实现接口还是一个类。

public void print() {

System.out.println("print");

}

}

public void method(){

//Inner i = new Inner();

//i.print();

//new Inner().print();

//Inter i = new Inner(); //父类引用指向子类对象

**new Inter() { //实现Inter接口，之后必须进行方法的重写。**

**public void print() { //重写抽象方法**

**System.out.println("print"); //主要是一气呵成的。**

**}**

**}.print(); //调用实现的抽象方法**

}

}

--------------------------------------------------------------------------------------------------------------------------

上边是写了两种方法，一种方法就是：

class Demo1\_NoNameInnerClass {

public static void main(String[] args) {

Outer o = new Outer();

o.method();

}

}

interface Inter {

public void print();

}

class Outer {

**new Inter() { //实现Inter接口，之后必须进行方法的重写。**

**public void print() { //重写抽象方法**

**System.out.println("print"); //主要是一气呵成的。**

**}**

**}.print(); //调用实现的抽象方法**

}

}

-------------------------------------------------------------------------

### 10.14\_面向对象(匿名内部类重写多个方法调用)

A:案例演示

匿名内部类的方法调用

---------------------------------------------------------------------------------------------------------------------------

class Demo2\_NoNameInnerClass {

public static void main(String[] args) {

Outer o = new Outer();

o.method();

}

}

interface Inter {

public void show1();

public void show2();

}

//匿名内部类只针对重写一个方法时候使用

class Outer {

public void method() {

/\*new Inter(){

public void show1() {

System.out.println("show1");

}

public void show2() {

System.out.println("show2");

}

}.show1();

Inter i = new Inter(){ //进行给到具体的对象，不匿名了！

public void show1() {

System.out.println("show1");

}

public void show2() {

System.out.println("show2");

}

/\*public void show3() {

System.out.println("show3");

}\*/

};

i.show1();

i.show2();

//i.show3(); //匿名内部类是不能向下转型的,因为没有子类类名

}

}

--------------------------------------------------------------------------------------------------------------------------

### 10.15\_面向对象(匿名内部类在开发中的应用)

--------------------------------------------------------------------------------------------------------------------------

ass Test1\_NoNameInnerClass {

public static void main(String[] args) {

//如何调用PersonDemo中的method方法呢?

PersonDemo pd = new PersonDemo ();

//pd.method(new Student()); 实例化之后进行传参。

pd.method(new Person() { //直接进行匿名内部类的写法。

public void show() {

System.out.println("show");

}

});

}

}

//这里写抽象类，接口都行

abstract class Person {

public abstract void show();

}

class PersonDemo {

//public void method(Person p) { //Person p = new Student(); 父类引用指向子类对象

/\*

Person p = new Person(){

public void show() {

System.out.println("show");

}

};

\*/

public void method(Person p) {

p.show();

}

}

class Student extends Person { //把接口进行实例化

public void show() {

System.out.println("show");

}

}

-------------------------------------------------------------------------------------------------------------

### 10.16\_面向对象(匿名内部类的面试题)

A:面试题

按照要求，补齐代码

interface Inter { void show(); }

class Outer { //补齐代码 }

class OuterDemo {

public static void main(String[] args) {

Outer.method().show();

}

}

要求在控制台输出”HelloWorld”

public static Inter method() {

return new Inter() {

public void show() {

System.out.println("HelloWorld");

}

};

}