# Java基础知识

## 一.简单Java程序

使用cmd进行目录跳转：命令cd

跳转盘符：直接打盘符+冒号、例如：d：

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

在你喜欢的地方新建一个"code"文件夹

使用cmd 跳转到你自己的code文件夹

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

写第一个java源程序

class HelloWorld{

public static void main(String[] args){

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

}

}

编译和运行java源程序

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

先编译：javac -- .java文件编译成.class文件

再运行：java -- 实际运行的是.class文件

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

配置jdk环境变量

完成：

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

扩展实际感受理解：显示三条信息在同一行里

## 二.Java数据类型

Java提供简单数据类型来表示整数、浮点数、字符以及布尔。这些类型称作基本类型。

整数：short，int，long

小数：float（单精度），double（双精度）

java有八种基本数据类型，boolean，char，byte

计算两个5之间的加减乘除和取余

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

5加上5 = 10

5减去5 = 0

5除以5 = 1

5乘以5 = 25

5的取余 = 0

用双引号包裹起来的东西叫做字符串

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

## 三.Java语言规范、API、JDK和IDE

Java是一个全面且功能强大的语言，可用于多种用途

1、Java有三个版本：

①Java标准版（Java Standard Edition、Java SE）

②Java企业版（Java EnterPrise Edition、Java EE）

③Java微型版（Java Micro Editon、Java ME）

2、JDK(Java Development Kit)

JDK用于开发和测试Java程序每个程序都是从命令行调用的

JRE(Java Runtime Environment)运行Java程序的工具。

## 四.一个简单的Java程序

1 public class Welcome{

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

3 //打印

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

5 }

6 }

#注意，这里的1~6是行号，为了让大家更方便观察及引用。真正敲代码时不需要敲入行号

解析：

第1行定义一个类。每个java程序至少应该有一个类。每个类都有一个名字。按照惯例，类名都是以大写字母开头。我这里取名为Welcome。

第2行定义了主方法、为了运行某个类，该类必须包含名字为main的方法。程序是从main方法开始执行的。

方法是包含语句的结构体，示例中的main方法包括了System.out.println语句。这句话在控制台输出打印“Hello World！”java中语句以分号（;）结束，称作语句结束符。

关键字

（上课讲解）

第三行是注释

①行注释 // 他可以注释一行

②块注释 /\*\*/ 注释多行

③…………

例子：

程序中的一对大括号将程序的一些组件组合，形成一个块。在Java中，每个块以左括号（{）开始，以右括号（}）结束。每个类都有一个将该类的数据和方法放在一起的类块。每个方法都有一个将该方法中的语句放在一起的方法块。块是可以嵌套的。

#简单说就是一个左括号必定有一个右括号，任何时候，你输入左括号的同时请不要忘记输入一个右括号来防止遗忘

#Java源程序是区分大小写的。

#任何语言都有语法，Java也是，你必须按照规定编写代码。如果违反了语法规则，比如忘记分号，编译器就会报告语法错误。

扩展实际感受理解：显示三条信息

进一步扩展：打印计算结果(11+4\*6)/(45-3.5)

#Java中乘法运算符是\* 除法运算符是/

以” ”包裹起来的东西，Java中称作字符串

## 五.创建、编译和执行Java程序

①使用任意记事本，我这里使用（Editplus）进行演示、创建Java源代码

#源文件扩展名必须是.java，并且文件名必须和类名完全相同。

例子中类名为Welcome、所以我新建一个文件名也是Welcome.java

②使用Java编译器将Java源文件编译成Java字节码文件。命令如下：

》javac Welcome.java

#在编译运行前需要安装和配置JDK、基础的DOS命令自行了解

Windows+r组合键 运行 输入cmd

cd 目录 代表跳转

D：代表跳转盘符

检验JDK是否安装：

1、假如你的jdk和环境变量配了的话，那么你打开CMD,然后输入javac

2、假如你没有配置环境变量，你可以在电脑里搜索关键字“javac”

这一过程示意：创建源代码（.java文件）→源代码存盘→编译源代码、生成

.class（字节码）文件→存盘→运行字节码→得到结果

#编译时如果没有语法错误，就会生成一个.class文件字节码。Java是高级语言，而字节码是低级语言类似机器指令（有兴趣的可以打开看看）它可以在任何带Java虚拟机（JVM）的平台运行 ← Java优点之一

③现在使用如下命令运行字节码：

java Welcome

练习 ：编写程序，显示“欢迎学习Java”、“欢迎来到计算机世界”和“编程真的有趣”

## 六.浅谈程序设计

抛出问题：如何计算一个圆面积？

↑↑↑

这不是问题、问题是如何编写程序计算圆的面积。

编写程序其实就干两件事！

>设计算法

>将算法翻译成程序代码

算法(algorithm)描述的是：如果要解决问题，所需要的动作以及这些动作执行的顺序。算法说穿了就是帮助我们在设计程序之前做一个规划。算法可以用自然语言或伪代码描述。

我这里举例子，比如说这个圆的面积问题：

1）读取半径

2）利用公式计算：面积=半径\*半径\*π

3）显示面积

那么当你编写代码时，就是要把算法翻译成程序。现在我给出详细代码因为每个程序都是以一个类的声明开始，所以以类名ComputeArea

public class ComputeArea{

//未完待续

}

之后因为每个程序都必须有一个main方法，所以理所当然程序扩展成：

public class ComputeArea{

public static void main(String[] args){

//1.读取半径

//2.计算圆的面积

//3.显示面积

}

}

但是这个程序需要读取用户输入的半径，产生了几个问题：

1）读取半径

2）将半径保存在程序中

为了解决第二个问题，我们需要声明一个叫做变量（variable）的符号。变量指定在程序中用于存储数据和计算结果的内存位置。每个变量都有自己的名字，可以用来访问它在内存的位置

#变量名应该尽量选择描述性的名字（descriptive name）而不是a，b这样的名字；那么这里半径毫无疑问可以选用radius表示，area表示面积。

但是还得让编译器知道radius之类的是什么东西，需要指明数据类型。

Java提供简单数据类型来表示整数、浮点数、字符以及布尔。这些类型称作基本类型。

整数：short，int，long

小数：float（单精度），double（双精度）

java有八种基本数据类型，boolean，char，byte

现在我把程序进一步扩展：

public class ComputeArea{

public static void main(String[] args){

double radius;

double area;

//1.读取半径

//2.计算圆的面积

//3.显示面积

}

}

这里，程序讲radius和area声明为变量。保留字double表明radius和area是以双精度浮点型保存在计算机中的。

保留字：java系统默认已经有了的

对于第一步，读取从键盘输入的数字不是简单的事情。所以就先给半径赋值一个固定的数。

之后完整的程序如下：

public class ComputeArea{

public static void main(String[] args){

double radius;//定义半径

double area;//定义面积

//给半径赋值

radius=20;

//计算面积

area=radius\*radius\*3.1415926;

//显示结果

System.out.println("半径为："+radius+

"的圆面积是"+area);

}

}

## 七.加号有两种意义

①是做加法

②是做字符串拼接，比如上放例子所示。如果两个操作数都是字符串，那么久拼接成一个字符串。如果其中一个不是字符串（例子中为double）那么就会先将这个double转换成字符串，在进行拼接。

提示：这个例子想说明的是，逐步开发的一种良好方法

#在代码中，字符串不能跨行。比如：

System.out.println("今天又上课了，我的心

好痛");

这样是肯定不对的，正确写法：

System.out.println("今天又上课了，我的心"+

"好痛");

\*从控制台读取输入

目的 ：为了能使用不同的半径。

可以使用Scanner类从控制台进行输入。

Java使用System.out表示标准输出设备、而用System.in来表示

标准输入设备。默认情况下，输出设备是显示器，输入设备是键盘

我们需要创建Scanner类这个对象,来读取System.in的输入

Scanner input = new Scanner(System.in);

解析：语法 new Scanner(System.in)表示创建了一个Scanner类型的

对象。语法Scanner input声明了input是一个Scanner类型的变量。

Input指向Scanner对象的引用

之后对象可以调用自己的方法。调用方法就是让这个对象完成某个任务

nextDouble()

使用Scanner时候必须先导包

import java.util.Scanner;

原因是java中 lang包是不用导的。

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

1、画个三角形(System.out.println())

2、给我求正方形的周长和面积。（需要用户手动输入正方形边长）

回家看视频第二章【变量和数据类型】和第三章【运算符】

必须掌握的重点内容：

1、常量，变量，数据类型

2、标准输入输出

3、运算符和表达式

===================================================================================

## 八.变量和数据类型

面试题：java的基本数据类型有几种、分别是什么？

8种：byte,char,short,int,long,float,double,boolean

整形：short,int,long

浮点型：float,double

布尔：true(真）、false（假）

byte字节、char字符

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

在方法里面的变量称作局部变量。

局部变量在使用之前必须初始化。

全局变量有默认值不需要初始化。

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

变量名需要遵守：如果变量名是由多个英文单词组成。第一个单词的首字母小写、第二个单词开始、首字母要大写。

例子：

superFirstName

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

常量

常量的值不会随着程序执行而发生变化，那么换句话说，常量意味着一个永不改变的永久数据。在之前的圆的例子中有π，这个π从逻辑上分析应该不是一个会改变的值。所以我们可以用常量来表示π。

格式：final 数据类型 常量名字 = 值;

例子：final double PI = 3.141;

#常量的命名必须全部大写,final关键字是声明常量的

常量的好处：

①不必重复输入同一个数

②可以提高程序的阅读性

简洁运算符

x = x+ 1;

x +=1 ;

#（+=）中间是不允许有空格的

x++;后置自增运算符，每次1

x--;后置自减运算符，每次1

++x;前置自增运算符，每次1

--x;前置自减运算符，每次1

#如果看到运算符在变量前面，那么变量先自身加1或减1，然后返回自身的值。同理，如果看到运算符在变量后面，那么变量同样自身加1或减1，但是返回的是原来的旧值。

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

## 九.运算符

逻辑运算符

&&逻辑与 ||逻辑或 ！非 &与 |或 ^异或

#我们平时编程时，最好选择逻辑与和逻辑或。因为逻辑与和逻辑或的效率大于普通的与和或。

原因：短路

#异或的运算非常复杂，但是有一个小技巧很实用：当你一个数异或两次，他的值不改变。具体应用在面试题：不用第三个变量交换两个数的值

位运算符

&

|

^

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

位运算

向左移<<、向左移动一个单位代表\*2

向右移>>、向右移动一个单位代表/2

#资深开发工程师在写代码运算时，多用位运算（左右移动）

具体用法：向左移一个单位代表\*2、向右移一个单位代表/2

————————————————————————————————————————————————————

>>> 无符号向右移

面试题：请用三种方式实现两个变量交换

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

面试题：

A：short i = i + 1;

B：short i += 1;

他们之间的区别：

① A式运行的结果编译出错，而B式运行后结果正常

② 因为A式采用的是赋合运算，导致 i+1计算完是int类型的，给short类型赋值一个int类型的时候，可能会超出short的范围，所以编译不通过。而B式采用的是简单运算符号，这种简单运算符号会自动把运算后的结果转换成左边的类型。

③ 从效率上分析，A式慢，B式快。

要从原理上分析：

A式这边总共执行了以下步骤：

1、先查找右边i的内存地址

2、计算i+1的值

3、再查找左边i的内存地址

4、赋值给i

B式这边总共执行了以下步骤：

1、查找i的内存地址

2、计算i+1的\*值

3、赋值给i

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

选择结构：

int a = 0;

//假设a等于0，则输出a是0

//假设a不等于0，则输出不存在a

## 十.if语句

### 1.单向if语句

if(布尔表达式){

语句(组);

}

-----------\*------------------------

if(radius>0){

area = radius\*radius\*3.14;

}

### 2.双向if语句

if(布尔表达式){

布尔表达式为真的时候语句(组);

}else{

布尔表达式为假的时候语句(组);

}

//从键盘输入一个数字，若大于0则显示他本身

//若等于0则显示"零"

//若小于0则显示他的绝对值

### 3.嵌套if语\*句

if(布尔表达式){

if(布尔表达式){

语句(组);

}

}

就是为了实现多重选择

他可以等价于、例子：学生成绩

int score = 0;

if(score >100){

class ="做了卷子但作弊";

}else{

if(score>90){

class = "优等生";

}\*

}

——————————————————————————————————、

if(score >100){

class ="做了卷子但作弊";

}else if(score >90){

class = "优等生";

}else {

}

这种写法更好

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

int a=0;

if(a>0)

System.out.println("haha");

这种写法是正确的888

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

练习题目：

1）首先产生两个随即整数number1,number2

2）判断number1是否大于number2，如果小于，交换两个数

3）问问题“number1-number2=？”

4）检查学生的答案并且显示答案是否正确

long number1 = System.currentTimeMillis()\*7%10;//获取当前时间的毫秒数

long number2 = System.currentTimeMillis()%10;

if(number1<number2){

long temp=0;

temp = number1;

number1 = number2;

number2 = temp;

}

System.out.println(number1+"-"+number2+"=？");

Scanner sc = new Scanner(System.in);

int answer = sc.nextInt();

if(answer == (number1-number2)){

System.out.println("正确");

}else{

System.out.println("不正确");

}

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

### 4.三目运算符

最简单的if...else...

if(a==0){

a=1;

}else{

a=-1

}

System.out.println((a==0)?1:-1);

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

## 十一.switch语句

switch语句可以用来有效的处理多重条件的问题，也就是当你if语句要频繁使用嵌套的时候，选择switch更好

switch括号里面填字符串是jdk1.7以后的新特性。

switch(char,int,short,byte){

case value1:

System.out.println("1");

break;

case value2:

System.out.println("2");

break;

default:

System.out.println("error");

break;

}

1）switch表达式必须计算出一个char,int,short,byte.并且必须用括号包住。

2）value1、value2…………必须与switch表达式的值有相同的类型、注意，这里的value都是常量表达式，也就是不能出现包含变量的例如x+1

3)当switch表达式的值与case的值相同，执行从此case开始的语句，直到遇到break或者达到末尾。

4）关键字break是可选的，break会立即终止整个switch语句。

5）default是默认的，他也是可选的。用来指定当没有条件符合时该怎么处理

6）case语句是顺序检查的，所以，case的出现顺序并不重要但是良好的编程习惯是根据逻辑顺序将case进行排列。

练习：

1.读取三条整数边，如果合法就判断能不能成为一个三角形，如果不合法就显示不合法，任意两条边长之和大于第三条边就视为合法。

2.输入一个整数，判断是否能被5或6整除，并且不能被5和6同时整除。

视频任务：第四章选择结构、第五章循环：

===================================================================================

## 十二.循环

### 1.while

格式：

while(布尔表达式){

语句(组);

}

例子：

int i=0;

while(i<10){

System.out.println("第"+i+"次");

i++;

}

程序员容易出错的地方：出一错误(off-by-one error)

错误例子(打印100次语句)：

int i = 0;

while(i<=100){//0<=100

System.out.println("xxx");

i++;

}

想无限循环怎么办:

while(true){

System.out.println("123");

}

注意：底下这种写法不好

while(1==1){

System.out.println("123");

}

#精度丢失扩展

例子：

double d = 1;double sum = 0;

while(d!=0){

sum += d;

d = d-0.1;

//System.out.println(d);

}

### 2.do-while

格式：

do{

语句(组)

}while(布尔表达式);

例子：

int i = 0;

do{

System.out.println("第"+i+"次");

}while(i<10);

#某些时候采用do-while会比while方便

### 3.for

格式：

for(初始条件;布尔表达式;每次循环后的操作){

语句(组);

}

例子：

int i=0;

while(i<10){

System.out.println("第"+i+"次");

i++;

}改写

for(int i=0;i<10;i++){

System.out.println("第"+i+"次");

}

#当循环次数确定以后，使用for

当循环次数不确定，使用while

#A:while(i<100){}

B:while(i<=99){}

i<100||i==99

i<=99不如i<100的效率高,所以我们建议使用A式的写法

举例：猜数字游戏

编写一个程序，产生一个随即整数1~100提示用户连续输入一个数字，直到匹配为止。对于用户，必须提示用户每次输入的值是太大还是太小。

Math.random();//随机产生一个double、范围是[0,1)

int answer = (int)(Math.random()\*100)+1;//0~99+1

Scanner sc = new Scanner(System.in);

while(true){

int input = sc.nextInt();

if(answer>input){

System.out.println("输入的值太大");

}else if(answer<input){

System.out.println("输入的值太小");

}else{

System.out.println("答对了");

break;

}

}

### 4.break和continue

break:跳出本循环

continue:结束本次循环执行下一次

goto 保留字

逻辑锻炼——练习：显示素数打印前N个素数，每行打印10个。

逻辑锻炼——练习：求最大公约数(greatest common divisor)用户输入两个正整数，然后找到它们的最大公约数。

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

### 5.foreach或者增强for循环

对比：

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

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

}

增强for：

for(int length:array){

System.out.println(length);

}

格式：

for(数据类型 变量名：源数据){

System.out.println(变量名);

}

#foreach循环不需要访问数组下标，来遍历整个数组。

但是，当想以其他顺序来进行遍历数组时。就不行了。

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

## 十三.数组（必须掌握一维数组、多维数组随意）

了解JVM的堆、栈

===================================================================================

### 1.静态方法

访问修饰符 static 返回值 方法名(参数){

}

public static void printPrime(int n){

}

public static void main(String[] args){

}

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

void 无返回值

### 2.数组

定义一个整形数组:

int[] arr = new int[20];

int[0] = 0;

int[1] = 1;

^^^^^

int[19] = 19;

//所有 new 关键字的对象都会放入堆里面

静态初始化数组：

int[] arr = {0,1,2,3,4,5,6,7,8,9};

java中数组写法：

int arr[]是C的、我们推荐的是int[] arr

循环、数组长度是已知的

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

int[i] = i;

}

### 3.数组的概念

一串连续的内存空间。

数组一旦创建、就长度不可变。

int[] arr = new int[20];

这里的arr是引用变量

arr = new int[50];

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

GC（垃圾回收器）

只要没有任何引用是一个垃圾，不定时的去回收垃圾

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

返回值必须满足所有的情况：

public class Test6{

public static void main(String[] args){

System.out.println(max(50,60));

}

public static int max(int n1,int n2){

if(n1>n2){

return n1;

}else if(n1<=n2){

return n2;

}

}

}

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

#方法的值传递：

值传递(pass-by-value)就是当调用带参数的方法时，实参的值传递给了形参。如果实参是变量而不是直接量，则将变量的值传递给形式参数。

#无论形参在方法中是否改变，该变量都不会受到影响。实参必须与方法签名中定义的形参次序和数量匹配，类型上兼容

public static void main(String[] args){

int i1 = 55;

int i2 = 555;

swap(i1,i2);

System.out.println("i1:"+i1+"i2"+i2);

}

public static void swap(int i1,int i2){

System.out.println("i1:"+i1+"i2"+i2);

int temp = i1;

i1 = i2;

i2 = temp;

System.out.println("i1:"+i1+"i2"+i2);

}

### 4.数组常用

1、打乱数组：在很多情况下都需要对数组进行重新排序。

称作打乱数组，我要现在实现这个功能，针对每个元素

array[i]，随即产生一个下标j，然后使得array[i]和

array[j]互换。

2、移动元素：在很多情况下，需要对数组进行向左或向右

移动，模拟元素向左移动一个位置，并且将第一个元素放到

最后一个位置。

3、冒泡排序

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

练习案例：我要统计每个字母出现的次数

1、随即产生100个小写字母，并将其放入字符数组中。

建立RandomCharacter类、

有getRandomLowerCaseLetter()方法.

他可以获取一个小写的随机字母。

2、对数组中每个字母进行出现的次数统计。

回家作业：

视频看第七章：类和对象、看完封装结束。

熟练掌握如何定义一个类模板。

理解封装，并能使用封装写类模板。

===================================================================================

求素数、求最大公约数、求最小公倍数

求2016年8月30号在2016年里是第几天

求n年m月d号在n年里是第几天

今天要掌握的内容：

1.方法（静态方法、实例方法、构造方法）

2.对象的使用（属性和方法）

3.会建立类模板（实体类）

4.学会使用封装

## 十四.类和对象

### 1.包

包主要就是用来放.java文件的，java建议我们最好将类放入包中。

而不要使用默认的包。使用包的时候,必须在程序首行写上语句：

package com.iotek.main;

### 2.对象

对象：

①特征(属性attribute)

②行为(方法method)

↓

类：

new 生成的东西都在堆里面

Cat cat = new Cat();创建对象，这里的cat是引用变量

语法：类名 对象引用变量 = new 类名();

使用访问操作符（.）访问对象方法和属性

#java有垃圾回收器（GC）,他会帮助你清理没有引用指向他的对象。

他是不定时的。

### 3.访问修饰符

有四种：public、protected、default、private

访问权限 本类 包 子类 其他包

public ∨ ∨ ∨ ∨

protected ∨ ∨ ∨ ×

default ∨ ∨ × ×

private ∨ × × ×

### 4.构造方法

特殊的方法、他有三个特征：

1、构造方法名必须和类名相同

2、没有返回值,void也不行

3、构造方法是你创建对象时调用的。new的时候调用的、他是

用来初始化对象

在你没有定义任何的构造方法时候，默认系统会产生一个无参的

构造方法。但是当你定义过其他任何一个构造方法时候，系统不会

生成无参的构造方法。

#建议每个实体类都有一个无参的构造方法

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

\*实例方法：

访问修饰符 返回值 方法名(参数列表){}

例：public void add(int aa){}

\*静态方法

访问修饰符 static 返回值 方法名(参数列表){}

例：public static void add(int aa){}

\*构造方法

访问修饰符 方法名(参数列表){}

例：public add(int aa){}

### 5.类和对象的概述

一、面向对象编程OOP的特点

结构化程序设计

\*程序=算法+数据结构

\*算法第一，数据结构第二

二、类的概念、定义

三、对象的概念

四、对象的三大特性

//五、类和对象的区别和联系

概念

\*类----抽象的概念

#类是创建对象的模板和蓝图

#类是一组类似对象的共同抽象定义

\*对象

#对象是类实例化的结果

#对象是实实在在的存在，代表现实世界的某一事物

\*对象的两个关键特性

#行为：对象能干什么

#状态：对象的属性，行为的结果

区别

\*类是静态定义

\*对象是动态实例

\*程序代码操作的是对象而非类

\*建模得到的是类而非对象

联系

\*类是对象的定义

\*对象的产生离不开类这个模板

\*类存在的目的是实例化得到对象

//六、定义类的步骤

#1)定义类名

#2)编写类的字段代表属性

#3)编写类的方法代表行为

#4)根据需要，考虑标志（通常从字段中确定）

七、类的建模是一个抽象和封装的过程

public class ConstructorDemo

{

public static void main(String [] args){

Person p=new Person("张三",10,"上海");

//1.在堆中开辟空间,给属性分配默认的初始值

//2.假设属性一开始就赋值了,就进行赋值工作.

//3.调用构造方法来对属性进行初始化

//p.setName("张三");

//p.setAge(10);

System.out.println(p.toString());

}

}

//当一个类没有显式声明一个构造方法的时候,系统会有一个默认的

//无参构造方法

class Person

{

private String name="李四";

private int age;

private String city;

public Person(){

System.out.println("无参数构造方法");

}

//带参数的构造方法

public Person(String name,int age,String city){

this();//必须放在首条语句

this.name=name;

this.age=age;

this.city=city;

}

public void setCity(String city){

this.city=city;

}

public String getCity(){

return city;

}

public void setName(String name){

this.name=name;

}

public String getName(){

return name;

}

public int getAge(){

return age;

}

public void setAge(int age){

this.age=age;

}

public String toString(){

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

return "名字:"+this.name+",今年:"+this.age+"岁,家住:"+this.city;

}

}

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

### 6.对象的创建和使用

一.引用数据类型的概念和特点

1)引用数据类型需实例化后才能使用

2)引用数据类型有自身的字段和方法

3)引用数据类型是对内存的指向，基本数据类型则直接访问到存放数据的内存

实例： Dog mydog = new Dog();

引用变量 对象

二.对象创建的过程

三.使用对象

1)成员访问符-----“.”

\*给对象属性赋值

#对象名.属性

\*对对象发送消息即调用方法

#对象名.方法（）

//方法定义和调用

\*方法分类

#返回值为void的方法

#带具体返回类型的方法

#不带参数的方法

#带参数的方法

\*方法返回值

#如果方法有返回值，方法中必须使用关键字return返回该值，返回值类型为该方法所定义的返回值类型

#强调：方法内可以调用其它的方法

public class MethodDemo

{

int number=30;

public void method1(){

System.out.println("返回类型为空且无参数的方法");

}

public String method2(){

return "带有具体返回类型且无参数的方法";

}

public void method3(String name){

System.out.println("返回类型为空且带有参数的方法,参数的值为"+name);

}

public String method4(String name,int age){

return "带有具体返回类型且带有多个参数的方法,参数的值为:"+name+","+age;

}

//传递基本数据类型的方法

public void operatorData(int age){

age++;

System.out.println(age);

}

//传递引用数据类型的方法

public void operatorArray(int[] array){

array[2]=100;

System.out.println(array[2]);

System.out.println("number:"+number);

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

}

public static void main(String [] args){

MethodDemo demo=new MethodDemo();

demo.method1();

String str=demo.method2();

System.out.println(str);

demo.method3("hello");

System.out.println(demo.method4("chenhao",30));

int age=10;

demo.operatorData(age);

System.out.println(age);

int [] array={1,2,3,4};

demo.operatorArray(array);

System.out.println(array[2]);

}

}

//String 对象的声明和操作

String s1="abcd";

String s2="abcd";

s1==s2----true

String s1=new String("abcd");

String s2=new String("abcd");

s1==s2----false

\*String 对象是不可变的

\*类中每一个看来会修改String 值的方法，其实都是创建了新的String 对象（包含修改后的字符串的内容）

\*字符串字面值存储于字符串中，String 对象优先指向该字符串池，避免发生生成重复的字符串实例

\*系统对String 的非修改处理效率很高，远远超过另外两个字符串StringBuffer和StringBuilder

\*String 对象的常用方法

#length（）--返回此字符串的长度

#charAt（int index）--返回指定索引处的char值

#concat（String str）--将指定字符串连接到此字符串的结尾

#contains（CharSequence s）--是否包含指定的字符串序列

#equals（Object anObject ）--将此字符串与指定的对象比较

#compareTo（String anotherString）--按字典顺序比较两个字符串

#indexOf(int ch)--返回指定字符串在字符串中第一次出现的索引

#split(String regex)--根据匹配给定的正则表达式来拆分此字符串

#subtring（int begainindex）--截取字符串

#toLower/upperCase（）--将指定的字符串进行大小写转换

public class StringDemo

{

public static void main(String [] args){

String content="Hello,My Friend,You are my best friend";

System.out.println(content.charAt(2));

System.out.println(content.compareTo("hello"));

//content=content.concat("I lied");

//System.out.println(content);

//content=content+" I Lied";

//System.out.println(content);

//System.out.println(content.endsWith("friend"));

//System.out.println(content.startsWith("Hell o"));

//System.out.println(content.contains("M y"));

/\*String s1="abc";

String s2="abc";

System.out.println(s1==s2);

System.out.println(s1.equals(s2));\*/

/\*String s1=new String("abc");

String s2=new String("abc");

System.out.println(s1==s2);

System.out.println(s1.equals(s2));\*/

//System.out.println(content.indexOf("o"));

//System.out.println(content.lastIndexOf("o"));

//System.out.println(content.indexOf("o",5));

//System.out.println(content.length());

//System.out.println(content.replace('e','a'));

/\*String [] arr=content.split(" ");

System.out.println(arr.length);

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

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

}\*/

//System.out.println(content.substring(5));

//System.out.println(content.substring(5,10));

//System.out.println(content.toLowerCase());

System.out.println(content.toUpperCase());

}

}

### 7.面向对象编程的优点

1)便于程序模拟现实世界中的实体

#用类封装建模实体对象的状态和行为

2)隐藏细节

#对象的行为和状态被封装在类中，外界通过公开的接口---调用类的方法来获取状态信息，不需关注内部细节如何实现

3)可重用

#可以通过类模板，创建多个对象实例，重用类的定义代码

五.Java对象的生命周期

1)创建

2)使用

3)清除

4)堆内存和栈内存

5)java堆内存主要存放由new创建的对象和数组

6)java栈内存主要存放局部变量、方法参数等

\*六.Java对象的内存管理

1)使用new创建对象，在堆内存分配对象空间、初始化

2)在方法栈中定义局部变量，持有对堆内存中对象的引用

3)方法执行完返回，栈内存自动释放，局部变量销毁

4)如果堆内存中对象没有变量引用它，成为垃圾，由垃圾回收器回收，释放所占堆内存

七.垃圾回收GC

1)垃圾回收器定期扫描内存，对于被使用的对象加上标记，按可能的路径扫描结束后清除未加标记的对象

public class Employee

{

public String name;

public int age;

//返回个人信息

public String toString(){

return "我的名字叫:"+name+",今年:"+age+"岁";

}

//在指定的字符串中查找相应的字符串出现的次数

/\*

src 朋友啊朋友,你是我最好的朋友

dst 朋友

indexOf("朋友");

\*/

public int countContent(String src,String dst){

int count=0;//计数器

int index=0;//保存找到朋友的下标

index=src.indexOf(dst);

//indexOf找到返回首次出现的下标,找不到就返回-1

while(index!=-1){

count++;//技术起加1

index+=dst.length();//指定从哪个下标往后找

index=src.indexOf(dst,index);

}

return count;

}

public static void main(String [] args){

Employee emp=new Employee();

emp.name="张三";

emp.age=30;

System.out.println(emp.toString());

String src="朋友啊朋友,你是我最好的朋友";

String dst="朋友";

int count=emp.countContent(src,dst);

System.out.println(dst+"出现的次数为:"+count+"次");

}

}

## 十五.封装,继承,多态

### 1.封装

①防止数据被篡改，避免数据被直接修改。应该使用private去将数据声明为私有的。

②满足有可能将会需要存储或读取数据的情况。提供一个get/set方法。

#通俗说，set方法访问器，get方法是修改器

判断以下程序是否正确：

public class Demo{

public static void main(String[] args){

private int a = 0;

System.out.println(a);

}

}

局部变量不能用private 和public protected等这些访问修饰符去修饰

import java.util.Date;

public class Test{

public static void main(String[] args){

Date date = null;

a(date);

System.out.println(date);

}

public static void a(Date date){

date = new Date();

}

}

练习：

使用Random类去创建种子为1000的对象，

使用nextInt(100)显示前50个随机整数。

封装概念

1)信息隐藏，隐藏对象的实现细节，不让外部直接访问到

2)将数据和方法包装进类中，加上具体实现的隐藏，共同被称作封装，其结果是一个同时带有特征和行为的数据类型

/\*

①防止数据被篡改，避免数据被直接修改。应该使用private去将数据声明为私有的。

②满足有可能将会需要存储或读取数据的情况。提供一个get/set方法。

#通俗说，set方法访问器，get方法是修改器

判断以下程序是否正确：

public class Demo{

public static void main(String[] args){

private int a = 0;

System.out.println(a);

}

}

局部变量不能用private 和public protected等这些访问修饰符去

修饰

import java.util.Date;

public class Test{

public static void main(String[] args){

Date date = null;

a(date);

System.out.println(date);

}

public static void a(Date date){

date = new Date();

}

}

练习：

使用Random类去创建种子为1000的对象，

使用nextInt(100)显示前50个随机整数。

对比全局变量：

1、局部变量需要手动去初始化，全局变量会默认初始化

2、全局变量的生命周期比局部变量长，局部变量只能活到方法结束为止。

常量：

public final int i = 3;

静态常量：

public static final int A = 3;

\*/

二.访问修饰符

1)信息隐藏是 OOP 最重要的功能之一，也是使用访问修饰符的原因

#有四种：public、protected、default、private

访问权限 本类 包 子类 其他包

public ∨ ∨ ∨ ∨

protected ∨ ∨ ∨ ×

default ∨ ∨ × ×

private ∨ × × ×

三.UML类图

#建议每个实体类都有一个无参的构造方法

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

\*实例方法：

访问修饰符 返回值 方法名(参数列表){}

例：public void add(int aa){}

\*静态方法

访问修饰符 static 返回值 方法名(参数列表){}

例：public static void add(int aa){}

\*构造方法

访问修饰符 方法名(参数列表){}

例：public add(int aa){}

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

一.字段封装

1)修改字段的可见性来限制对字段的访问

2)为每个字段创建一对赋值(setter)方法和取值(getter) 方法，用于公开对这些字段的访问接口

3)在setter和getter方法中，根据需要加入对字段操作的限制

二.字段封装要点

1)除非必须公开底层实现细节，否则应该将所有字段指定为private加以封装

2)使用字段封装，通过增加数据访问限制，增强了类的可维护性

三.方法封装

1)隐藏方法实现细节（方法体），向外部提供公开接口（方法头），以供安全调用

2)简化调用，方便修改维护

3)根据需要，可以私有化方法以供类内部使用---帮助方法helper

四.类封装

1)在任何相互关系中，具有关系所涉及的各方都遵守的边界，创建类就建立了与客户端(应用)程序员之间的关系

2)封装及访问控制首要存在原因：隐藏细节

3)封装及访问控制第二个原因：隔离变化，利于维护

对比全局变量：

1、局部变量需要手动去初始化，全局变量会默认初始化

2、全局变量的生命周期比局部变量长，局部变量只能活到方法结束为止。

常量：

public final int i = 3;

静态常量：

public static final int A = 3;

===================================================================================

this关键字的两种常用方法

①指代对象，例如：this.a = a;这里的this是指调用本类实例的对象

②构造方法调用同一个类的另一个构造方法。当你使用this去调用其他构造

方法的时候你必须放在第一句。

super的用法可以参照this

### 2.继承

关键字：extends

1、继承的子类不是父类的子集、而比父类更具体更大。

2、静态方法不能被重写，如果你在子类新写了这个静态方法。叫做重新定义

3、Object是所有类的基类，所有类默认都会继承于Object。

4、Java只支持单继承，不支持多继承。但可以多重继承

5、子类不能继承父类的构造方法。

6、在new子类的对象时，子类的构造方法会调用父类的构造方法。构造方法链

方法的重写和重载：

重写：子类重写父类，用@Override表示重写，方法名一样，参数列表也要相同、返回值类型不能变，访问修饰符只能变大或不变。

重载：方法名必须相同，参数列表不同，返回值，访问修饰符无关

继承is-a关系：我们继承，必须要考虑父类和子类存在关系的。不要仅仅只是为了重复使用而去继承。

has-a：

### 3多态

动态绑定：

Dog dog = new Dog();

Object obj = new Dog();

System.out.println(obj.toString());

假设Dog类继承了Object、那么执行这里的某一个方法的时候，会依次从Dog类开始往父类找起，一直找到Object类。

instanceof比较对象

作业：视频内容，第七章剩下的。抽象类和接口

面试题：抽象类和接口的区别。

===================================================================================

super关键字

this-------当前对象的引用

super------父类对象的引用

特点

\*super和this关键字的特点类似：super代表的是父类对象的引用

\*当子父类的成员出现同名时，可以通过super来进行区分

\*子类的构造方法中，通过super关键字调用父类的构造方法

public class JavaTeacher extends Teacher{

public JavaTeacher(String name,String school){

super(name,school);

}

}

通过调用父类的构造方法，完成对相关字段值的初始化

强调：当构造一个子类对象的时候一定会先调用父类的构造方法来构造父类的对象。调用父类构造方法的语句必须是子类构造方法中的第一条指令。

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

public class TeacherDemo {

public static void main (String[]args){

B b=new B();

b.showB();

}

}

class A{//父类

public A(){

System.out.println("a的构造方法");//父类先输出

}

public void showA(){

System.out.println("A");

}

}

class B extends A{//子类，继承父类

public B(){

super();//调用父类无参数的构造方法，可以省略

System.out.println("b的构造方法");

}

public void showB(){

System.out.println("B");

super.showA();//调用父类里的方法

}

}

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

public class TeacherDemo{

public static void main(String[]args){

JavaTeacher javaTeacher=new JavaTeacher ("张三年","安科技");

javaTeacher.teachering();

DBTeacher dbTeacher=new DBTeacher

("李斯","监督局");

dbTeacher.teachering();

}

}

class Teacher{

private String name;

private String school;

public Teacher(String name,String school){

this.name=name;

this.school=school;

}

public void teachering(){

System.out.println("授课前准备");

}

}

class JavaTeacher extends Teacher{

public JavaTeacher(String name,Stirng school){

super(name,school);

}

public void teachering(){

super.teachering();

//teachering.teachering();陷入死循环

//super.teachering();继承父类

System.out.println("打开eclipse");

System.out.println("书写java代码");

}

}

class DBTeacher extends Teacher{

public DBTeacher(String name,Stirng school){

super(name,school);

}

public void teachering(){

System.out.println("打开Oracle");

System.out.println("书写p1-sp1指令");

}

}

一.多态性是指“多种行为”

二.同样的方法调用而执行不同操作、运行不同代码

三.多态通过分离做什么和怎么做，从另一个角度将接口和实现进行分离

1)“封装”通过合并特征和行为来创建新的数据类型

2)“实现隐藏”通过将细节“私有化”把接口和实现进行分离

四.多态消除了类型之间的耦合关系

public class AnimalDemo{

public static void main(String []args){

//父类的引用变量可以引用其子类的对象

Animal animal1=new Dog("旺财");//向上转型

//向上转型首先是安全的,但可能会导致子类方法的丢失

//父类的引用变量只能调用父类中有的方法或子类重写父类的方法

animal1.eat();

//animal1.sleep();

Animal animal2=new Cat("招财猫");

animal2.eat();

//向下转型是不安全的

if(animal2 instanceof Cat){

System.out.println("aa");

Cat cat=(Cat)animal2;

cat.sleep();

}

}

}

class Animal

{

private String name;

public Animal(String name){

this.name=name;

}

//这是一个通用的方法,通用的方法实现没有太大的意义,

//只是告诉其子类去实现它

public void eat(){

}

}

class Dog extends Animal{

public Dog(String name){

super(name);

}

//对父类的方法进行重写

public void eat(){

System.out.println("啃骨头");

}

}

class Cat extends Animal{

public Cat(String name){

super(name);

}

//对父类的方法进行重写

public void eat(){

System.out.println("吃鱼肉");

}

public void sleep(){

System.out.println("睡觉");

}

}

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

对象上下转型

五.由子类转型成父类，在继承图上是向上移动的，一般称为向上转型

六.向上转型是从一个较专用类型向较通用类型转换，所以总是安全的，也就是说，子类是父类的一个超集

七.向上转型过程中，类接口中唯一可能发生的事情是丢失方法，而不是获取方法

八.与之相反的操作是向下转型，不安全（可能需要instanceof操作符协助）

九.LSP(liskov替换原则)：子类型必须能够替换掉它们的基类型

十.安全的上转型和LSP的实施，充分体现继承的“is-a”关系

十一.多态的概念基于对象引用的动态绑定特性，动态绑定是多态现象的根源

十二.多态实现过程：

1)子类重写父类的方法

2)代码中向父类型变量发出消息（静态绑定）

3)运行时，根据变量实际引用的对象类型决定调用哪个方法（动态绑定）

十三.静态绑定在编译期进行

十四.动态绑定在运行期进行

十五.方法重载解析根据静态绑定

十六.方法改写解析根据动态绑定

十七.开闭原则

1)软件实体应当对扩展开放，对修改封闭

十八.遵循开闭原则设计出的模块具有两个主要特征：

1)对于扩展是开放的（Open for extension）。意味着模块的行为是可以扩展的。当应用需求改变时，可以对模块进行扩展，使其具有满足那些改变的新行为。即可以改变模块的功能

2)对于修改是封闭的（Closed for modification）。对模块行为进行扩展时，不必改动模块的源代码或者二进制代码。模块的二进制可执行版本无需改动

十九.instanceof 运算符用来在运行时通过返回一个布尔值来指出对象是否是特定类或者是它的子类的一个实例

result = object instanceof class

result：布尔类型

object：必选项，任意对象表达式

class：必选项，任意已定义的对象类

◆如果 object 是 class或其子类的一个实例，则 instanceof 运算符返回 true，如果不是或者object 是 null，则返回 false

◆在对对象做下转型之前，没有其它有关对象类型信息时务必使用instanceof 判断一下，以避免抛出ClassCastException异常

二十.多态应用

◆简单来说，多态是具有表现多种形为能力的特征

◆同一个实现接口，使用不同的实例而执行不同操作

public class PrinterDemo{

public static void main(String [] args){

ColorPrinter cp=new ColorPrinter("惠普");

BlackPrinter bp=new BlackPrinter("戴尔");

ZhenPrinter zp=new ZhenPrinter("戴尔");

School school=new School();

//school.setColorPrinter(cp);

school.setPrinter(zp);

school.print("hello,java");

}

}

class Printer{

private String brand;

public Printer(String brand){

this.brand=brand;

}

public String getBrand(){

return brand;

}

//打印的方法应该由其子类来具体的实现.

public void print(String content){

}

}

//开闭原则:对修改是封闭的,对扩展是开放的.不要违反开闭原则.

//可以使用多态解决这个问题.父类的引用变量可以引用其子类的对象

class School{

/\*private ColorPrinter cp=null;

private BlackPrinter bp=null;

private ZhenPrinter zp=null;\*/

private Printer p=null;//安装打印机

//安装彩色打印机

/\*public void setColorPrinter(ColorPrinter cp){

this.cp=cp;

}

//安装黑白打印机

public void setBlackPrinter(BlackPrinter bp){

this.bp=bp;

}

public void setZhenPrinter(ZhenPrinter bp){

this.zp=zp;

}\*/

//拿父类的引用变量作为参数,好处就是可以接受任何其子类的对象

//越是抽象的东西代表越稳定.

public void setPrinter(Printer p){

this.p=p;

}

public void print(String content){

//交给中心所安装的彩色打印机来打印

p.print(content);

}

}

class ColorPrinter extends Printer{

public ColorPrinter(String brand){

super(brand);

}

//对父类的方法进行重写

public void print(String content){

System.out.println(getBrand()+"彩色打印:"+content);

}

}

class BlackPrinter extends Printer{

public BlackPrinter(String brand){

super(brand);

}

//对父类的方法进行重写

public void print(String content){

System.out.println(getBrand()+"黑白打印:"+content);

}

}

class ZhenPrinter extends Printer{

public ZhenPrinter(String brand){

super(brand);

}

//对父类的方法进行重写

public void print(String content){

System.out.println(getBrand()+"针式打印:"+content);

}

}

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

方法重写--override

什么是方法重写

\*方法重写是指子类可以根据需要对从父类继承来的方法进行改写，是多态机制的前奏。

方法重写的注意点

\*重写方法必须和被重写方法具有相同的方法名称，参数列表和返回值

\*重写方法不能使用比被重写方法更严格的访问权限。（public private）

\*父类中的私有方法不能被重写。（private 私有）

\*在子类重写的方法中继承调用父类，父类被重写的方法可以通过调用（super.函数名）获取。

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

final关键字

特点

\*final可以用来修饰变量、方法、类。

\*final修饰的变量是一个常量。一旦赋了值就不能再修改。（常量一般都和static关键字配合使用）

\*final修饰类代表此类不能被继承。

\*final修饰方法代表此方法不能被重写。

public class FinalDemo{

public static void main (String[]args){

A a=new A();

a.showNumber1();

B a=new B();

b.showNumber2();

}

}

//常量类是不能被继承的

/\*final\*/class A{

public final int number1=1;//常量

public int number2=2;

public void showNumber1(){

//number1++;常量是不能被修改的

System.out.println(number1);

}

public /\*final\*/ void showNumber2(){

//加final，子类不能对父类中的常量方法进行改写（重写）

System.out.println(number2);

}

}

class B extends A{

public void showNumber2(){

System.out.println(number2);

}

}

===================================================================================

实例方法：

静态方法：

构造方法：

final:

1、final可以修饰方法、变量、类

在某些时候，你有可能想防止类被继承。只需要使用final去修饰类

例如：Math

修饰方法使得方法不能被重写

修饰变量使得变量不能被修改

修饰类使得类不能被继承

#final可以修饰局部变量

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

## 十六.抽象类和接口

### 1.抽象类

关键字：abstract

定义：一个类抽象到它没有任何具体的实例、叫做抽象类.abstract修饰类表示这个类是抽象类，修饰方法表示是抽象方法。

1.抽象类可以不全是抽象方法，甚至可以没有抽象方法。

2.抽象方法不能用final和static修饰

3.子类去继承一个非抽象类的父类、子类也可以声明为抽象类

4.有些时候想让子类不实现父类的抽象方法而选择继续抽象下去。

5.子类可以重写父类的方法让他定义为抽象方法

一.抽象方法

1)只有方法头没有方法体的方法称为抽象方法

2)抽象方法用abstract来修饰

3)抽象方法代表一种不确定的操作或行为

4)抽象方法不能被调用

二.抽象类

1)定义中含有抽象方法的类叫做抽象类

2)抽象类用abstract来修饰

3)抽象类代表一种抽象的对象类型

4)抽象类不能实例化

5)抽象类中可以有具体方法，可以没有抽象方法

一.设计模式（Design pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了可重用代码、让代码更容易被他人理解、

保证代码可靠性

二.模板方法模式

1)定义：一个模板方法用一些抽象的操作定义一个算法，而子类将重定义这些操作以提供具体行为

2)意图： 定义了在一个操作中的一个算法框架，把一些步骤推迟到子类去实现。模板方法模式让子类不需要改变算法结构而重新定义特定的算法步骤

一.抽象方法

1)只有方法头没有方法体的方法称为抽象方法

2)抽象方法用abstract来修饰

3)抽象方法代表一种不确定的操作或行为

4)抽象方法不能被调用

二.抽象类

1)定义中含有抽象方法的类叫做抽象类

2)抽象类用abstract来修饰

3)抽象类代表一种抽象的对象类型

4)抽象类不能实例化

5)抽象类中可以有具体方法，可以没有抽象方法

三.设计模式（Design pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性

四.模板方法模式

1)定义：一个模板方法用一些抽象的操作定义一个算法，而子类将重定义这些操作以提供具体行为

2)意图： 定义了在一个操作中的一个算法框架，把一些步骤推迟到子类去实现。模板方法模式让子类不需要改变算法结构而重新定义特定的算法步骤

一.接口概念和作用

1)Java接口是对类的需求描述

2)Java接口表示：所有实现该接口的类看起来都像这样

3)接口被用来建立类与类之间的协议

4)通过实现接口，java类可模拟多重继承

二.接口定义

1)Java接口中只有公开的抽象方法和静态常量字段

2)interface关键字用于定义接口类型

3)implements关键字用于类实现接口

三.接口与抽象类的区别

1)抽象类和具体实现类之间是一种继承关系，采用抽象类的方式则父类和子类在概念上应该是相同的

2)接口和实现类在概念上不要求相同，接口只是抽取相互之间没有关系的类的共同特征，而不去关注类之间的关系，它可以使没有层次关系的类具有相同的行为

3)抽象类是对一组具有相同属性和行为的逻辑上有关系的事物的一种抽象，而接口则是对一组具有相同属性和行为的逻辑上不相关的事物的一种抽象

4)对于接口和抽象类的选择，反映出设计人员看待问题的不同角度。抽象类用于一组相关的事物，表示的是“is-a”的关系；而接口用于一组不相关的事物，表示的是“like-a”的关系

### 2.接口

关键字：interface

定义：接口的结构和类相似，只包含抽象方法和常量。

实现接口的关键字：implements

1.接口不能new实例化对象

2.接口可以多实现

3.接口只允许出现常量

4.接口可以继承接口

练习：编写一个方法，返回对象数组中最大的对象

public static Object max(Comparable[] a)

所有对象都是Comparable接口的实例。对象在数组中的顺序由compareTo方法决定。

写测试类，测试 ：

1.由5个字符串组成的数组

2.由5个日期(java.util.Date)构成的数组

面试题：抽象类和接口的区别：

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

#浅谈策略模式

人去餐厅吃饭、餐厅里有日式午餐，中式午餐，西式午餐

具体的抽象对象：他就是接口去实现

public interface IStrategy{

//怎么吃

void howToEat();

}

具体策略对象：实现不同的算法

public class JapanLunch implements IStrategy{

@Override

public void howToEat(){

System.out.println("吃日本");

}

}

public class CLunch implements IStrategy{

@Override

public void howToEat(){

System.out.println("吃美食");

}

}

public class ALunch implements IStrategy{

@Override

public void howToEat(){

System.out.println("吃西式午餐");

}

}

这是一个环境对象：他持有了接口的引用

public class Context{

private IStrategy strategy;

public Context(IStrategy strategy){

this.strategy = strategy;

}

public void howToEat(){

strategy.howToEat();

}

}

public class Test{

public static void main(String[] args){

Context context;

//吃日本

context = new Context(new JapanLunch());

context.howToEat();

//吃中餐

context = new Context(new CLunch());

context.howToEat();

//吃西餐

context = new Context(new ALunch());

context.howToEat();

}

}

1.首先需要把一个类中经常使用的东西单独提取出来，写成一个接口

2.然后需要有一个类去持有接口的引用（Context）

3.根据运行的时候实例的变化去调用这个接口的类行为

class A{

public void a(){}

}

class B{

public void a(){}

}

class C extends A,B{

}

interface A{

void a();

}

interface B{

void a();

}

class C implements A,B{

@Override

public void a(){

}

}

看视频：第八章

了解四种内部类。

☆掌握匿名内部类。

常用类

☆String、StringBuffer和StringBuilder区别

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

Object类常见方法

String toString()

\*返回该对象的字符串描述性信息。默认输出的格式是：类名[字段值，字段值....]；

\*只要对象与一个字符串通过“+”连接，系统就会自动调用toString()以获得对象的字符串描述；

\*常被改写：可以根据用户的需要对其进行重写。

boolean equals()

\*Object类原始功能是实现判断两个是否具有相同的引用；要求判断两个对象状态的相等性；

\*在jdk标准库中提供了一些类，比如前面所讲的String，后续索要讲的Date。他们都对 equals方法进行了重写；

\*常被改写：可以根据用户的需要对其进行重写。

public class ObjectDemo{

public static void main (String[]args){

Student stu1=new Student("jack",20);

System.out.println(stu1.toString());

Student stu2=new Student("jack",20);

System.out.println(stu1.toString(stu2));

}

}

class Student /\*extends Object\*/{

private String name;

private int age;

public Student(String name,int age){

this.name=name;

this.age=age;

}

public String toString(){

return "name"+name+"age"+age;

}

//重写了equals方法

public boolean equals(Object obj){

if(this==obj){ //自己跟自己比

return true;

}

if(obj==null){

return false;

}

if (this.getclass()!=obj.getclass()){

return false;

}

Student stu=(Student)obj;

if(this.age!=stu.age){

return false;

}

if(this.name==null){

if (stu.name!=null){

return false;

}

}else if(!this.name.equals(stu.name)){

//把this.name与stu.name比较

return false;

}

return true;

}

}

## 十七.内部类

### 1.成员内部类

(实例内部类)

二.特性:

◆不能有静态成员(字段，方法)

◆内部类对象外部创建语法:outter.new Inner()

◆实例方法访问外部类实例成员:Outter.this.成员

### 2.静态内部类

1)内部类之前修饰static关键字

2)特性:

◆可以有实例，静态成员(字段,方法)

◆实例方法中，无法访问外部实例成员

### 3.匿名内部类

一.匿名内部类概念

匿名类是没有名称的类，所以没办法引用它们。必须在创建时，作为new语句的一部分来声明它们。这就要采用另一种形式的new语句：

二.匿名内部类语法

new interfacename(or superclass) { //方法重写 }

当某个方法的参数是接口或者抽象类时，调用这个方法时除了可以直接new一个接口的实现类或者抽象类的子类之外,还可以使用匿名内部类的语法实现临时方法重写(只用一次)，这种用法在Java中非常常见，尤其在后期Android课程中会频繁使用，务必熟练掌握

三、匿名内部类

格式：new 父类或接口(){

//类体

}

当匿名内部类访问局部变量的时候，必须给局部变量加final修饰符、

jdk1.8里面他会自动给你加上。

1、传递一份拷贝的数据成员去保证生命周期一致。

2、final用来保证原局部变量的值始终不变。

扩展：

jdk1.8新特性：

支持lambda表达式，他可以用来替换匿名内部类原有的写法，简化匿名内部类

Lambda表达式语法结构：

(参数1，参数2，参数3……参数n)->{

//重写方法

}

你的接口中只有一个抽象方法需要去重写、jdk1.8版本

### 4.局部内部类

一.局部内部类特点：

1）定义在代码块、方法体内、作用域（使用花括号“{}”括起来的一段代码）内的类叫局部内部类。

2）局部内部类访问外部类的属性和方法使用“外部类名.this.属性名”和“外部类名.this.方法名(参数)”的形式。

3）对外部世界完全隐藏，只能在其作用域内生成对象。

二.局部内部类有以下限制：

1）局部类不能加访问修饰符，因为它们不是类成员。

2）成员内部类不能与外部类重名。

3）局部内部类访问作用域内的局部变量，该局部变量需要使用final修饰

## 十八.String、StringBuffer和StringBulider

### 1.String字符串

1.String是常量,不可变的类。如果你要修改String,就会产生一个新的String对象。

然后在堆里面开辟新空间导致效率会慢。还会影响GC去频繁的垃圾回收。

2.String s = "abc";这种写法会将abc放入常量池，以后如果出现String s2="abc";

这种样子的写法会去优先查找常量池中是否存在已有字符串，如果有就直接引用，反之、

加入常量池。

3.String s = "A"+"AB"+"AA"+"www";

JVM有一个特殊机制,就是说他会认为以上这种写法的字符串结果是

String s = "AABAAwww";

String str = "A";

String str1 = "B";

String s = str+str1;

不能引用对象,如果你引用了对象，JVM不认识

### 2.StringBuffer和StringBulider都是动态字符串

1.StringBuilder单线程，不安全. StringBuffer多线程，安全，效率比StringBulider稍微慢一点

2.效率高，不用频繁的创建新对象，直接可以进行修改。

3.使用append()方法来追加字符串

包装类

1.包装类就是java在设计时与8个基本数据类型对应的8个类

比如：int---Integer long---Long

2.包装类用来方便涉及对象的操作

3.自从拆装箱(jdk1.5以上)

Integer i = new Integer(10);

Integer i1 = 10;//装箱

int j = i1;//拆箱

4.integer里面有一个缓存区，他会缓存-128~127之间的整数。当调用valueOf(int i)

的时候，如果你的值在缓存区之内，就直接从缓存区拿出来对象。反之，就生成新对象。

看视频：异常+泛型

泛型你只要看得懂，用的出就可以

1.请写一个方法，要求去掉字符串收尾两端空格

2.我有一个学生类（姓名，年龄，分数）、请分别使用Comparable和Comparator

来对学生进行排序（从大到小排，按照年龄，如果年龄一样按照分数排序）

3.现在有如下格式的一串文本内容：

<h学号tm=\l1><t.\*.姓名.d>..小黑.<t/h><a分数./..bc>

<ab\c><t\h></td>80</ht\ml>

<h学号tm=\l2><t.\*.姓名.d>..小红.<t/h><a分数./..bc>

<ab\c><t\h></td>50</ht\ml>

<h学号tm=\l3><t.\*.姓名.d>..小白.<t/h><a分数./..bc>

<ab\c><t\h></td>10</ht\ml>

取出里面的姓名学号分数，创建出一个学生对象。

===================================================================================

/\*

有个Person类，包括姓、名、性别、年龄属性：现在有个需求，需要对Person进行以下规则的排序

\* 谁的姓拼音靠前，谁就排前面。

\* 然后对名字进行排序。如果同名，女性排前头。

\* 如果名字和性别都相同，年龄小的排前头。

\*

int compare(Object o1, Object o2) 返回一个基本类型的整型

如果要按照升序排序，

则o1 小于o2，返回-1（负数），相等返回0，01大于02返回1（正数）

如果要按照降序排序

则o1 小于o2，返回1（正数），相等返回0，01大于02返回-1（负数）

\*/

Person类

package com.ibm.test;

public class Person {

String firstname,lastname;

char sex;

Integer age;

public Person(String firstname, String lastname, char sex, Integer age) {

this.firstname = firstname;

this.lastname = lastname;

this.sex = sex;

this.age = age;

}

public Integer getAge() {

return age;

}

public void setAge(Integer age) {

this.age = age;

}

public String getFirstname() {

return firstname;

}

public void setFirstname(String firstname) {

this.firstname = firstname;

}

public String getLastname() {

return lastname;

}

public void setLastname(String lastname) {

this.lastname = lastname;

}

public char getSex() {

return sex;

}

public void setSex(char sex) {

this.sex = sex;

}

}

Comparators 类：

package com.ibm.test;

/\*

\* 谁的姓拼音靠前，谁就排前面。

\* 然后对名字进行排序。如果同名，女性排前头。

\* 如果名字和性别都相同，年龄小的排前头。

\*

int compare(Object o1, Object o2) 返回一个基本类型的整型

如果要按照升序排序，

则o1 小于o2，返回-1（负数），相等返回0，01大于02返回1（正数）

如果要按照降序排序

则o1 小于o2，返回1（正数），相等返回0，01大于02返回-1（负数）

\*/

public class Comparators {

public static java.util.Comparator getComparator() {

return new java.util.Comparator() {

public int compare(Object o1, Object o2) {

if (o1 instanceof String) {

return compare( (String) o1, (String) o2);

}else if (o1 instanceof Integer) {

return compare( (Integer) o1, (Integer) o2);

}else if (o1 instanceof Person) {

return compare( (Person) o1, (Person) o2);

}else {

System.err.println("未找到合适的比较器");

return 1;

}

}

public int compare(String o1, String o2) {

String s1 = (String) o1;

String s2 = (String) o2;

/\*

\* System.out.println("s1=="+s1+" s2=="+s2);

\* 取出数组相邻的两个姓firstname或者名lastname

\*/

int len1 = s1.length();

int len2 = s2.length();

int n = Math.min(len1, len2);

char v1[] = s1.toCharArray();

char v2[] = s2.toCharArray();

int pos = 0;

while (n-- != 0) {

char c1 = v1[pos];

char c2 = v2[pos];

if (c1 != c2) {

return c1 - c2;

}

pos++;

}

return len1 - len2;

}

public int compare(Integer o1, Integer o2) {

int val1 = o1.intValue();

int val2 = o2.intValue();

/\*

\* System.out.println("val1=="+val1+" val2=="+val2);

\* 取出数组相邻的两个人的年龄age

\*/

return (val1 < val2 ? -1 : (val1 == val2 ? 0 : 1));

}

public int compare(char o1, char o2) {

/\*

\* System.out.println("o1=="+o1+" o2=="+o2);

\* 取出数组相邻的两个人的性别

\* 对return的逻辑解释：

\*

比如两个人的性别分别是男和女，当第一个人的性别是‘男’，则返回1

（返回1的对于compare方法来说，即认为o1（男）>（女）o2，而o1>o2返回1则按照升序排序，升序排序即将小的（女）排在前面 )

还不懂的话：接着看compare方法的解释

int compare(Object o1, Object o2) 返回一个基本类型的整型

如果要按照升序排序，

则o1 小于o2，返回-1（负数），相等返回0，01大于02返回1（正数）

如果要按照降序排序

则o1 小于o2，返回1（正数），相等返回0，01大于02返回-1（负数）

\*/

return ((o1==o2)? 0 : (o1=='男'?1:-1));

}

public int compare(Person o1, Person o2) {

String firstname1 = o1.getFirstname();

String firstname2 = o2.getFirstname();

String lastname1 = o1.getLastname();

String lastname2 = o2.getLastname();

char sex1 = o1.getSex();

char sex2 = o2.getSex();

Integer age1 = o1.getAge();

Integer age2 = o2.getAge();

/\*return 逻辑解释如下：

\* 谁的姓拼音firstname靠前，谁就排前面。

\* 然后对名字lastname进行排序。如果同名，

\* 女性sex==false排前头。（sex的值true表示男性，false表示女性）

\* 如果名字和性别都相同，age年龄小的排前头。

\*/

return (compare(firstname1, firstname2) == 0 ?

(compare(lastname1, lastname2) == 0 ?(compare(sex1, sex2) == 0 ? (compare(age1, age2) == 0 ? 0 :

compare(age1, age2)) :

compare(sex1, sex2)) :

compare(lastname1, lastname2)) :

compare(firstname1, firstname2));

}

};

}

}

Main方法

package com.ibm.test;

import java.util.Arrays;

/\*

\* 谁的姓拼音靠前，谁就排前面。

\* 然后对名字进行排序。如果同名，女性排前头。

\* 如果名字和性别都相同，年龄小的排前头。

\*

\*/

public class Main {

public Main() {

}

public static void main(String[] args) {

Person[] person = new Person[] {

new Person("ouyang", "feng", '男', new Integer(27)),

new Person("zhuang", "gw", '男', new Integer(27)),

new Person("deng", "jx", '男', new Integer(28)),

new Person("deng", "jx", '女', new Integer(27)),

};

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

System.out.println("排序前=" + person[i].getFirstname()+" "+person[i].getLastname()+" "+person[i].getAge()+" "+person[i].getSex());

}

/\*

\* 根据sort方法的第一个参数是person，然后在Comparators.getComparator()方法中找到对应的方法

\* compare(Person o1, Person o2)，参数与person类型一致

\*/

java.util.Arrays.sort(person, Comparators.getComparator());

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

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

System.out.println("排序后=" + person[i].getFirstname()+" "+person[i].getLastname()+" "+person[i].getAge()+" "+person[i].getSex());

}

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

Integer[] data = new Integer[]{2,3,1};

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

System.out.println("排序前=" + data[i]+" ");

}

Arrays.sort(data,Comparators.getComparator());

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

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

System.out.println("排序后=" + data[i]+" ");

}

}

}

运行效果：

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

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Calendar;

import java.util.Date;

public class test16 {

/\*\*

\* @param args

\* @throws ParseException

\*/

public static void main(String[] args) throws ParseException {

// TODO Auto-generated method stub

SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

Date d1=sdf.parse("2012-09-08 10:10:10");

Date d2=sdf.parse("2012-09-15 00:00:00");

System.out.println(daysBetween(d1,d2));

System.out.println(daysBetween("2012-09-08 10:10:10","2012-09-15 00:00:00"));

}

/\*\*

\* 计算两个日期之间相差的天数

\* @param smdate 较小的时间

\* @param bdate 较大的时间

\* @return 相差天数

\* @throws ParseException

\*/

public static int daysBetween(Date smdate,Date bdate) throws ParseException {

SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");

smdate=sdf.parse(sdf.format(smdate));

bdate=sdf.parse(sdf.format(bdate));

Calendar cal = Calendar.getInstance();

cal.setTime(smdate);

long time1 = cal.getTimeInMillis();

cal.setTime(bdate);

long time2 = cal.getTimeInMillis();

long between\_days=(time2-time1)/(1000\*3600\*24);

return Integer.parseInt(String.valueOf(between\_days));

}

/\*\*

\*字符串的日期格式的计算

\*/

public static int daysBetween(String smdate,String bdate) throws ParseException{

SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");

Calendar cal = Calendar.getInstance();

cal.setTime(sdf.parse(smdate));

long time1 = cal.getTimeInMillis();

cal.setTime(sdf.parse(bdate));

long time2 = cal.getTimeInMillis();

long between\_days=(time2-time1)/(1000\*3600\*24);

return Integer.parseInt(String.valueOf(between\_days));

}

}

===================================================================================

Comparable

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

/\*\*

\* 两个类要想比较大小，

\* 就必须实现Comparable<T>接口，类型参数T通常设为实现该接口的类名

\* 实现Comparable<T>接口后，还要覆盖public int compareTo(<T> object)方法

\* 如果两个对象相等返回0

\* 当前对象大于方法传过来的对象时，返回一个正整数

\* 当前对象小于方法传过来的对象时，返回一个负整数

\* 而且相等时必须和equals()方法保持一致，即eaquls()方法要返回true

\* 否则相反

\*/

public class comparable Point implements Comparable<comparablePoint>{

public int x;

public int y;

public comparablePoint(int x,int y){

this.x=x;

this.y=y;

}

public int compareTo(comparablePoint point) {

int s1=x\*x+y\*y;

int s2=point.x-point.x+point.y\*point.y;

return s1-s2;

}

public String toString(){

return "["+x+","+y+"]";

}

public static void main(String args[]){

if(new comparablePoint(3, 4).compareTo(new comparablePoint(1, 1))>0)

System.out.println("true");

//线性表

List<comparablePoint> points=new ArrayList<comparablePoint>();

points.add(new comparablePoint(1, 6));

points.add(new comparablePoint(3, 4));

points.add(new comparablePoint(5, 2));

System.out.println(points);

Collections.sort(points);//调用Collections工具类进行排序

System.out.println("排序后的情况");

System.out.println(points);

}

}

结果

[plain] view plain copy

在CODE上查看代码片派生到我的代码片

true

[[1,6], [3,4], [5,2]]

排序后的情况

[[3,4], [5,2], [1,6]]

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

1.IComparable接口

namespace System{

public interface IComparable{

//Less than zero This instance is less than obj.

//Zero This instance is equal to obj.

//Greater than zero This instance is greater than obj.

int CompareTo(object obj);

}

}

2.一个example

namespace ConsoleApplicationCompare{

class Program{

static void Main(string[] args){

Student s1 = new Student();

Student s2 = new Student();

s1.Name = "张三";

s2.Name = "张三";

s1.Score = 112;

s2.Score = 211;

Console.WriteLine(((IComparable)s1).CompareTo(s2));

Console.ReadKey();

}

}

public class Student: IComparable{

public string Name { set; get; }

public int Score { set; get; }

public int CompareTo(object obj) {

int result = 0;

Student o = (Student)obj;

if (this.Score > o.Score)

result = 1;

else if (this.Score == o.Score)

result = 0;

else

result = -1;

return result;

}

}

}

3.IComparable<T>接口

namespace System{

// Type parameters: T: The type of objects to compare. That is, you can use either the type you specified or any type that is less derived.

public interface IComparable<in T>{

int CompareTo(T other);

}

}

4. example 第二版

namespace ConsoleApplicationCompareT{

class Program{

static void Main(string[] args){

Student s1 = new Student();

Student s2 = new Student();

s1.Name = "张三";

s2.Name = "张三";

s1.Score = 112;

s2.Score = 211;

Console.WriteLine(s1.CompareTo(s2));

Console.ReadKey();

}

}

public class Student : IComparable<Student> {

public string Name { set; get; }

public int Score { set; get; }

public int CompareTo(Student obj){

int result = 0;

if (this.Score > obj.Score)

result = 1;

else if (this.Score == obj.Score)

result = 0;

else

result = -1;

return result;

}

}

}

===================================================================================

## 十九.异常

### 1.超类是Throwable

他有两个子类：Error和Exception

其中Exception他还有个子类为RuntimeException

①Error系统的错误，如果你出现了Error，立马去查看你是不是有很愚蠢的错误。如果

没有低级错误，请尽可能的通知用户终止程序,因为你什么也干不了。

②Exception：代表的是由一些外部环境和程序引起的错误。使得能够捕获和处理这些

错误

③RuntimeException：运行时异常，比如除数为0，数组越界，空指针异常

### 2.异常处理模型

模型基于三种操作：

①声明异常 throws关键字

②抛出异常 throw

③捕获异常 try catch块

try{

//需要捕获代码

}catch(异常1 e1){

}catch(异常2 e2){

}catch(异常3 e3){

}

异常3必须不能小于异常2和1

异常2必须不能小于异常1

### 3.finally子句

无论什么情况都会执行的子句。try{}catch(Exception e){}finally{}

使用finally子句的时候，catch可以省略不写，而且finally中不建议使用return

练习：在一个类中编写一个方法，这个方法搜索一个字符数组中是否存在某个字符，

如果存在，则返回这个字符在数组中第一次出现的位置（下标），否则，返回-1.

要搜索的字符数组和字符都以参数形式传递给该方法。如果传入的数组为null，就抛出IllegalArgumentException异常。

===================================================================================

## 二十.泛型

动机：从jdk1.5以后出现的

Java去使用一个泛型的类、接口、方法

jdk1.5前Comparable：

public interface Comparable{

public int compareTo(Object o);

}

jdk1.5以后泛型：

public interface Comparable<T>{

public int compareTo(T o);

}

好处：

Comparable c = new Date();

c.compareTo("aaa");

Comparable<Date> c = new Date();

c.compareTo("aaa");

调用compareTo的时候啊，是Date类型和字符串进行比较。这样代码肯定会出错，

因为使用了泛型，所以你传递的那个参数必须是Date类型，那么会产生编译错误。

由于这个错误你不用在运行时被检测到了，所以用泛型更可靠。

二、List集合

1.ArrayList底层是数组，不同步

①数组是不可变的，ArrayList是一个动态的数组。

②数组可以放基本数据类型，而ArrayList不能放基本数据类型，但是能放null

③Vector和ArrayList一样，但是同步

2.LinkedList底层是链表，不同步

3.凡是实现了iterable接口的集合都可以使用foreach进行遍历

①LinkedList插入和删除数据的效率比ArrayList高。

②ArrayList因为底层是数组，所以查找非常迅速，比LinkedList快

③对于List，我们都可以将集合变成iterator去进行遍历

④她们都可以添加重复的元素，有序。

练习：

1.描述java集合框架。列出接口，便利抽象类和具体类。

2.给出答案，LinkedList是否包含ArrayList所有方法、

哪些方法在LinkedList中有，但在ArrayList中没有。

3.编写一个程序，用户输入1的时候，对这个集合进行排序(sort)，用户输入2的时候，

对集合进行打乱顺序(Shuffle)，用户输入3的时候，颠倒顺序（Reverse）。

(链表)

——————————————————————————————————————————————————————————————————————————

4.List对象中存放多个Person对象(此对象包含，名字，年龄、id)。

按Person的年龄

从小到大排序，假设年龄相等的话再按名字的大小来排序。求出年龄最大的那个学生信息。

=================================================================================

public static void main (String[]args){

String[] strs={};

AccountUitl.printdata(AccountUitl.account(strs));

}

class AccountUitl{

public static Map<String,Intrger>account(String[]strs){

Map<String,Integer>data=new HashMap<String,Integer>();

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

String str=strs[i];

if(data.get(str)==null){

data.put(str,1);

}else{

data.put(str,data.get(str)+1);

}

}

return data;

}

public static void printData(Map<String,Integer>data){

Set<Entry<String,Integer>>entrys=data.entrySet();

for(Entry<String,Integer>entry:entrys){

System.out.println(entry.getKey()+"出现的次数为"+entry.);

}

}

}

## 二十一.枚举

一.装箱(Boxing):

1)基本数据类型?引用类型

二.拆箱(UnBoxing):

2)引用类型?基本数据类型

一.枚举把相关的常量分组到一个枚

举类型里，而且枚举提供了比常量更多的方法

二.枚举声明

public enum Color {

RED, GREEN, BLACK, YELLOW

}

三.枚举变量定义和使用

Color colorEnum = Color.BLACK;

System.out.println(colorEnum);

一.了解三个类的异同点

二.在特定需求下使用哪一个类实现功能。

一.枚举把相关的常量分组到一个枚

举类型里，而且枚举提供了比常量更多的方法

二.枚举声明

public enum Color {

RED, GREEN, BLACK, YELLOW

}

三.枚举变量定义和使用

Color colorEnum = Color.BLACK;

System.out.println(colorEnum);

一.示例 – Date、SimpleDateForma

二.会获取当前日期，时间，按指定格式，如”2010-8-9 18:30:00”

四、枚举类型

1.是从jdk1.5以后出现的新类型，关键字enum.

2.定义静态常量：public static final、定义枚举就是为了简便我们写常量。

枚举可以将相关常量写在一个枚举类里，常常和switch case配合使用。

3.枚举可以有构造方法，必须是私有的。常量里面例如RED("哈哈",10)

4.枚举可以有实例方法、也可以有静态方法，变量等.

5.枚举都继承了java.lang.Enum，所以依据java单继承他不能再继承别的类了，但是可以

实现多个接口。

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

## 二十二.多线程

### 1.线程的概念

线程（thread）是指一个从头到尾的一个任务（执行流程）。

并发：同一时间点上、只有一个处理器执行任务，在多个线程之间快速切换。

并行：同一时间点，有多个处理器执行任务。干2件以上的事情。

### 2.线程的生命周期：

new(创建)——Runnable(就绪)——(running)运行——(Bolcked)阻塞——(dead)死亡

守护线程（Daemon Thread）：

特点：前台线程全部死亡后才会死亡，专门就是用来服务前台线程。

### 3.使用多线程

Java内置了多线程功能，提供我们写代码。

任务就是对象，创建对象必须先有类。类必须实现Runnable接口或者继承Thread类。

它只有一个run()方法。你需要实现这个方法来告诉程序这个线程我想如何执行。

sleep():休眠，回到阻塞状态

yield():回到就绪状态

### 4.线程创建和启动方式

①使用Thread类进行创建和启动。它将任务和运行任务的机制混在了一起，将任务从线程中

分离出来是较好的设计理念。所以不推荐使用这种方法。

②实现Runnable接口进行创建和启动。

③线程池，Java提供Executor接口执行线程任务，ExecutorService接口管理任务。

newFixedThreadPool(int)创建的是固定数量的线程

newCachedThreadPool()创建与线程数量匹配的缓存区

#线程中的run方法表示如何完成这个操作，JVM通过start方法开启线程自动调用run方法。如果你直接调用run方法，那么不开启线程只执行你所调用的那个run方法。

#信号量

尝试：假设现在有个厕所，然后这个厕所有三个坑，现在有五个人同时上厕所。模拟上厕所实战。要求打印出这个结果：A进了坑、B进了坑、C进了坑，D，E等待A、B、C出来了，D、E进去。

线程同步：

synchronized关键字

练习：创建200个线程，定义一个类叫Account模拟账户，一个叫AddMoney的类来添加账户金额(每次添加1元)。每个线程都向同一个账户添加一元.创建这些线程并启动运行。

加强：创建启动两个线程，一个用来向账户中存款，另一个取款。当取款数量大于余额时，取款线程等待，不管什么时候，只要向账户新存入钱。存储线程去通知取款线程重新取款，如果余额仍未达到数额，取款线程继续等待存款。

wait()

notify()

Java从1.5后，出现了lock对象，来控制线程通信协作，需要用newCondition方法

await()等待

signal()唤醒

package com.iotek.demo;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class AccountTest{

public static Account account = new Account();

public static void main(String[] args){

ExecutorService executor = Executors.newCachedThreadPool();

for(int i=0;i<200;i++){

executor.execute(new AddMoney(account));

}

executor.shutdown();

while(!executor.isTerminated());//判断线程全部执行完毕

System.out.println("余额"+account.getMoney());

}

}

class AddMoney implements Runnable{

Account account;

public AddMoney(Account account){

this.account = account;

}

@Override

public void run() {

synchronized(account){

account.addMoney(1);//new 肯定不对

}

}

}

class Account{

private int money=0;

public int getMoney(){

return money;

}

public void addMoney(int i){

int newMoney = money + i;

try {

Thread.sleep(6);

} catch (InterruptedException e) {

e.printStackTrace();

}

money = newMoney;

}

}

视频IO流：必须要看,字节流中：掌握FileInputStream和

FileOutPutStream。其他一般掌握。

字符流：BufferedReader和BufferedWriter

## 二十三.数据库

create table student(

id integer,

name varchar(50),

age int

);

查看数据库：show databases；

建立数据库：create database databasename;

调用数据库：use databasename；

建立表：create table tablename(

属性1 变量名1，属性2 变量名2，........

);

查看表：show tables;

查看属性：desc tablename;

增：insert into tablename(col1,col2...........（属性）) values(values1,values2,values3...........（变量）);

//insert into tablename;

//values(values1,values2,values3...........);

查看全部：select values1,values2,values3...... from tablename ;

查看某一行：select values1,values2,values3...... from tablename where col1=values1;

两个表查看：select s.id,s.name,s.age,sc.score from student s,score sc where s.id=sc.id;

改：update tablename set col1=values1,col2=values2,.........where col1=values2;

删：delete from tablename where col1 is values;

退出数据库：quit/exit;

删表：drop table tablename;

删数据库：drop database databasename;

student

id name age

1

2

score

id score

1 80

2 70

student

id name age score

1 80

2 70

select s.id,s.name,s.age,sc.score from student s,score sc where s.id=sc.id;

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

数据库约束：

主键 自增 非空

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

放入jar包：

1、新建lib文件夹

2、将mysql的jar包放进lib文件夹

3、右键 Build Path -》 add to Build Path

JDBC:

1、加载JDBC驱动程序：Class.forName();

2、提供JDBC连接的URL：URL

3、创建数据库的连接：DriverManager

4、创建一个Statement：PreparedStatement

5、执行SQL语句：

①增删改executeUpdate

②查executeQuery

6、处理结果：

①增删改 返回int

②查ResultSet

7、关闭JDBC对象：

全部都close();

create table users(

id integer primary key auto\_increment,

name varchar(50),

password varchar(50));