0006：入门

程序：计算机直行某些操作或解决某个问题而编写的一系列有序指令的集合。

//①老师写的最简单的程序，完成1+1

Public class Test {

Public static void main(String[] arges) {

int res = 1 + 1:

//显示

System.out.printIn(“结果=” + res):

}

}

使用cmd命令可执行此任务。

0007 版本的选择

目前最常用的版本是8,11(支持时间长)。

java技术体系

0008 java语言特性

Java特性

1.Java语言是面向对象的(oop)

2.Java的强类型机制，异常处理、垃圾的自动收集等是Java程序健壮性的重要保证。

3.Java语言是跨平台型的。

编写的一个Java文件不受平台影响，即一个编译好的.class文件可在多个平台运行。

4.Java语言是解释性语言

常见的解释型语言：Java, javascript

常见的编译型语言：c/c++

两者的区别是: 解释型语言所编译好的代码需要解释器执行后才能被机器执行，编译型语言编译好的代码可直接被机器执行。

0009 编译器的选择

先选用文本编辑器本sublime，后续再使用IDEA和Eclipse。

0010 Java运行机制及运行过程

1. Java核心机制-Java虚拟机(java virtual machine)

⑴ JVM是一个虚拟的计算机，具有指令集并使用不同的存储区域。负责执行指令，管理数据、内存、寄存器，包含在JDK中。

⑵对于不同的平台，有不同的虚拟机。

⑶Java虚拟机制屏蔽了底层运行平台的差异，实现了“一次编译，到处运行”

JVM

运行Java

Test.class

编译(javac)

Test.java

1. JDK基本介绍

⑴ JDK的全称是JAVA Development Kit,java(开发工具包)

JDK=JRE+java的开发工具[java,javac,javadoc,javap等]

⑵JDK十提供给Java开发人员使用的，包含Java的开发工具，也包含JRE，所以安装了JDK，就不用再安装JRDE了。

1. JRE基本介绍

⑴JRE的全称是 Java Runtine Enviroment Java运行环境

JRE=JVM+Java SE标准类库(Java的核心类库[类])

⑵如果想要运行一个开发好的Java程序，计算机中只需要安装JRE即可。

0013 配置环境变量

为了在任何一个目录下都可以直行java和javac，所以需要配置环境变量。

步骤：

⑴高级系统设置环境变量

⑵增加JAVA HOME 环境变量，指向jdk 安装目录

⑶编辑python 环境变量，增加 %JAVA\_HOME%\bin

⑷打开DOS命令执行，任意目录下敲入javac/java，如果出现javac 的参数信息，配置成功。

0014 java开发步骤

用户环境变量与系统环境变量区别：在用户环境变量配置只针对此用户，在系统环境变量配置使用更广泛。

//1.public class Hello 表示Hello是一个类，是一个public公有的类

//2.Hello{}表示一个类的开始和结束

//3.public static void main(string[] args)表示一个主方法，即我们程序的入口

// main()表示方法的开始和结束

//System.out.printIn("hello,word~");表示输出"hello,word~"到屏幕

pulic class Hello {

//编写一个main方法

public static void main(string[] args) {

System.out.printIn("hello,word~");

}

}

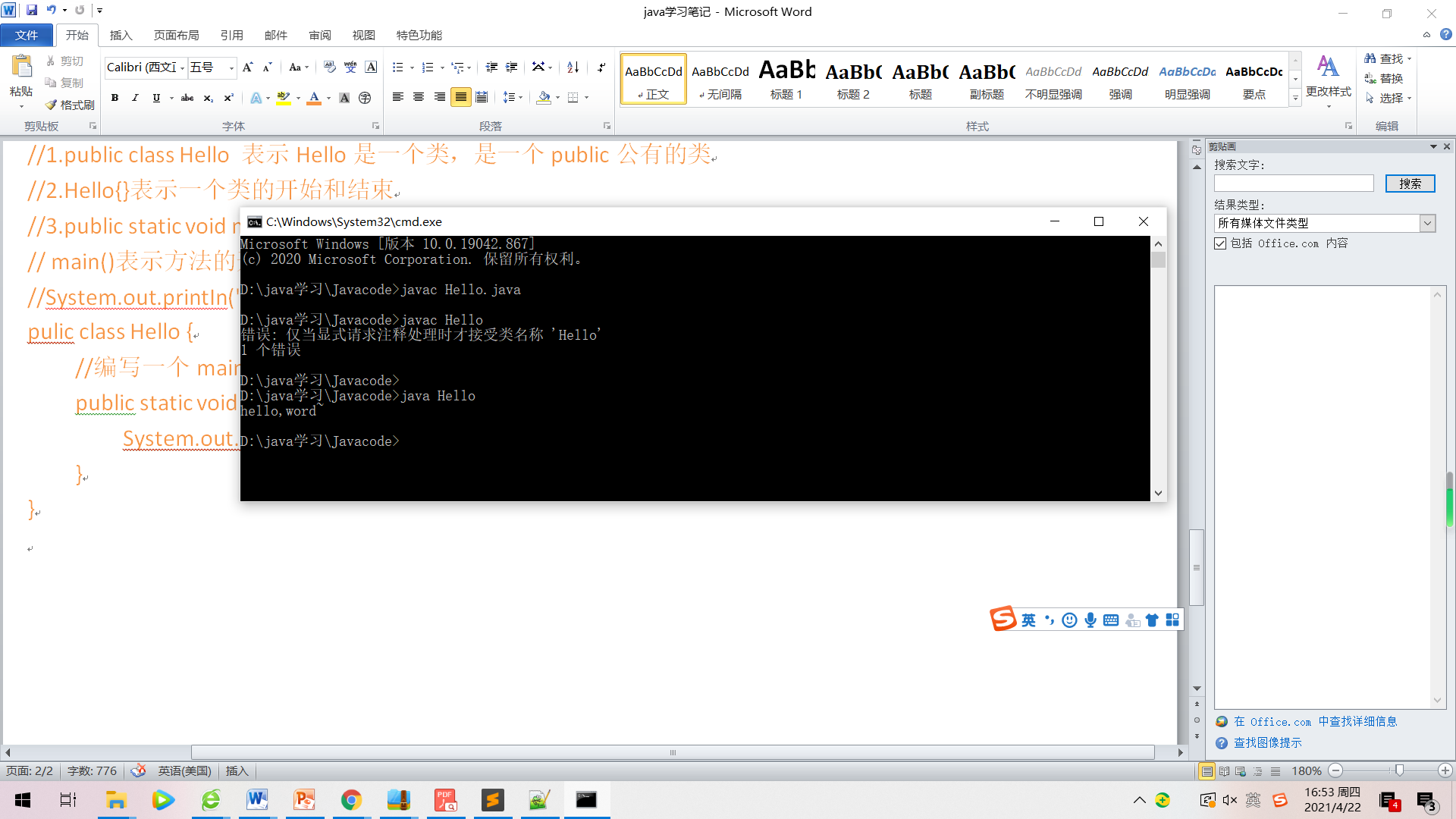
一个简单的java开发步骤：

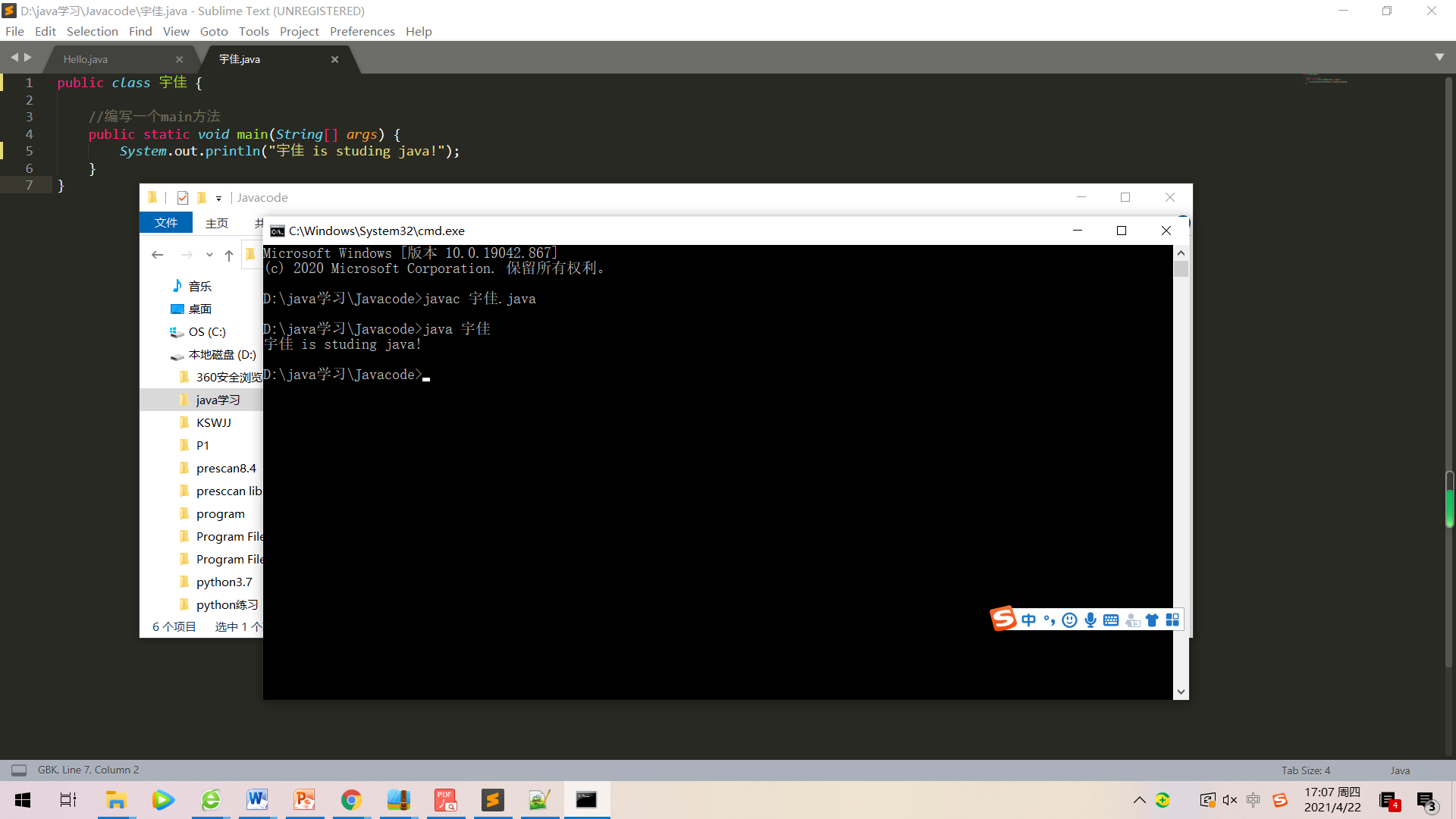
⑴将Java代码编写到扩展名为.java的文件中

⑵通过javac命令对该Java文件进行编译，生成.class文件

⑶通过java命令对生成的生成.class文件进行运行注意不写后缀

文件有中文时，要转换成GBK格式。





作业一截图

0017 java执行流程分析

javac编译

字节码文件

源文件

结果

Java运行

.class文件

.java文件

对源文件进行修改，需重新进行编译才能输出新的内容。

0018 Java开发注意事项和细节说明

⑴Java源文件以.Java为扩展名。源文件的组成部分是类（class）

⑵Java应用程序的执行入口是main()方法,它有固定的书写格式：

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

⑶Java语言严格区分大小写

⑷Java由一条条语句构成，Java每条语句以“；”结束

⑸大括号需成对出现（习惯先写{}再写代码）

⑹一个源文件最多只能有一个public类。其他类的个数不限

⑺编译后，每一个类对应一个.class

⑻如源文件包含一个public类，则文件名以该类命名。

⑼可将main方法写在非public类中，然后指定运行非public类，这样入口方法就是非public的main方法。

开发的基本步骤：

⑴将JAVA代码编写到扩展名为.java的文件中

⑵通过javac命令对该Java文件进行编译，生成.class文件

⑶通过Java命令对生成的class文件进行运行

0019 如何快速学习

基本原理，语法快速入门研究细节及如何优化

0020 Java转移字符

在控制台，输入TAB可快速补全文件名

⑴\t:制表位，实现对其功能

⑵\n：换行输出

⑶\：转义字符

⑷\r:一个回车

public class Changechar {

//编写一个main方法

public static void main(String[] args) {

System.out.println("北京\t天津\t上海");

System.out.println("北京\n天津\n上海");

System.out.println("\\北京\n天津\n上海");

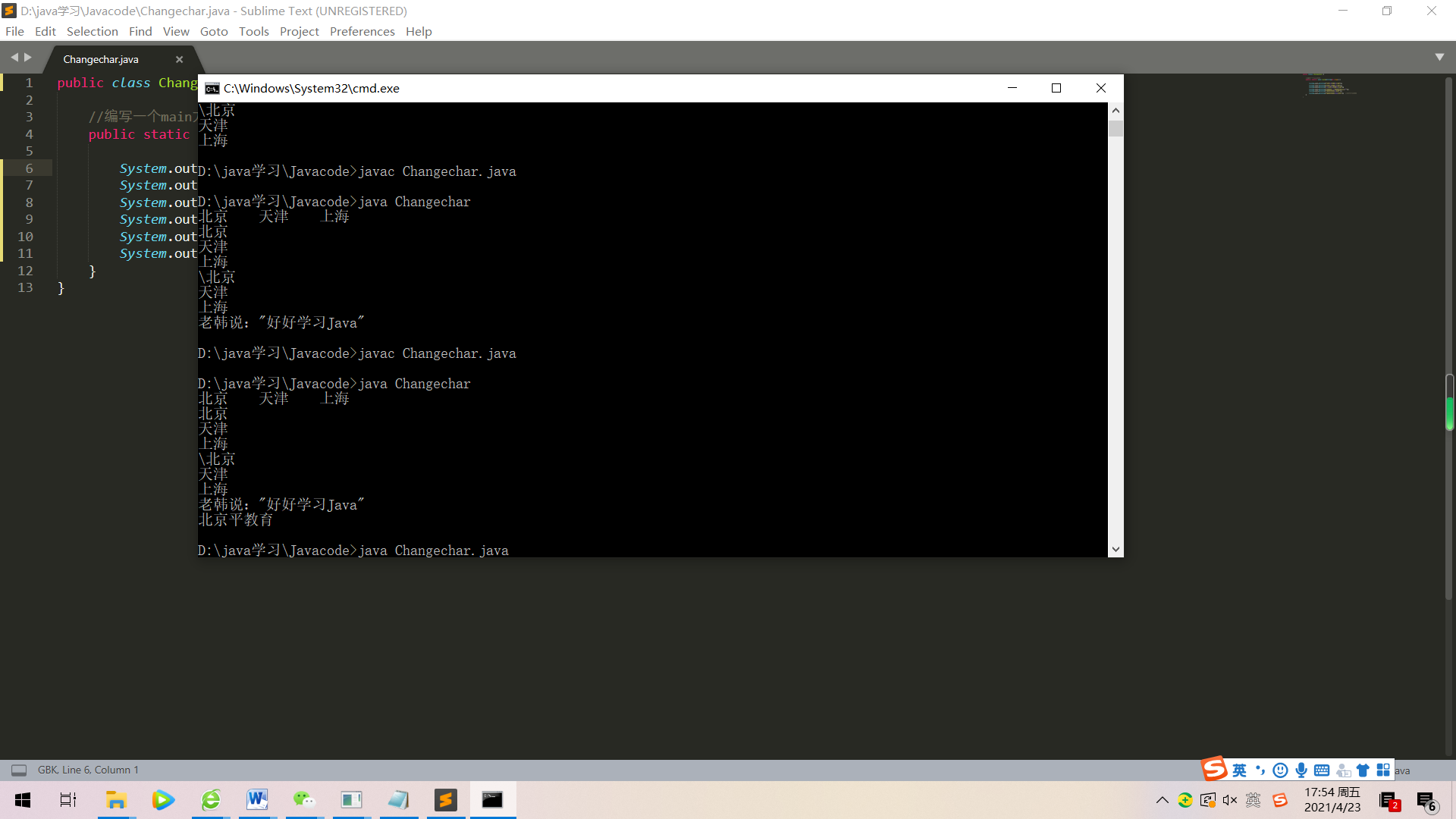
System.out.println("老韩说：\"好好学习Java\"");

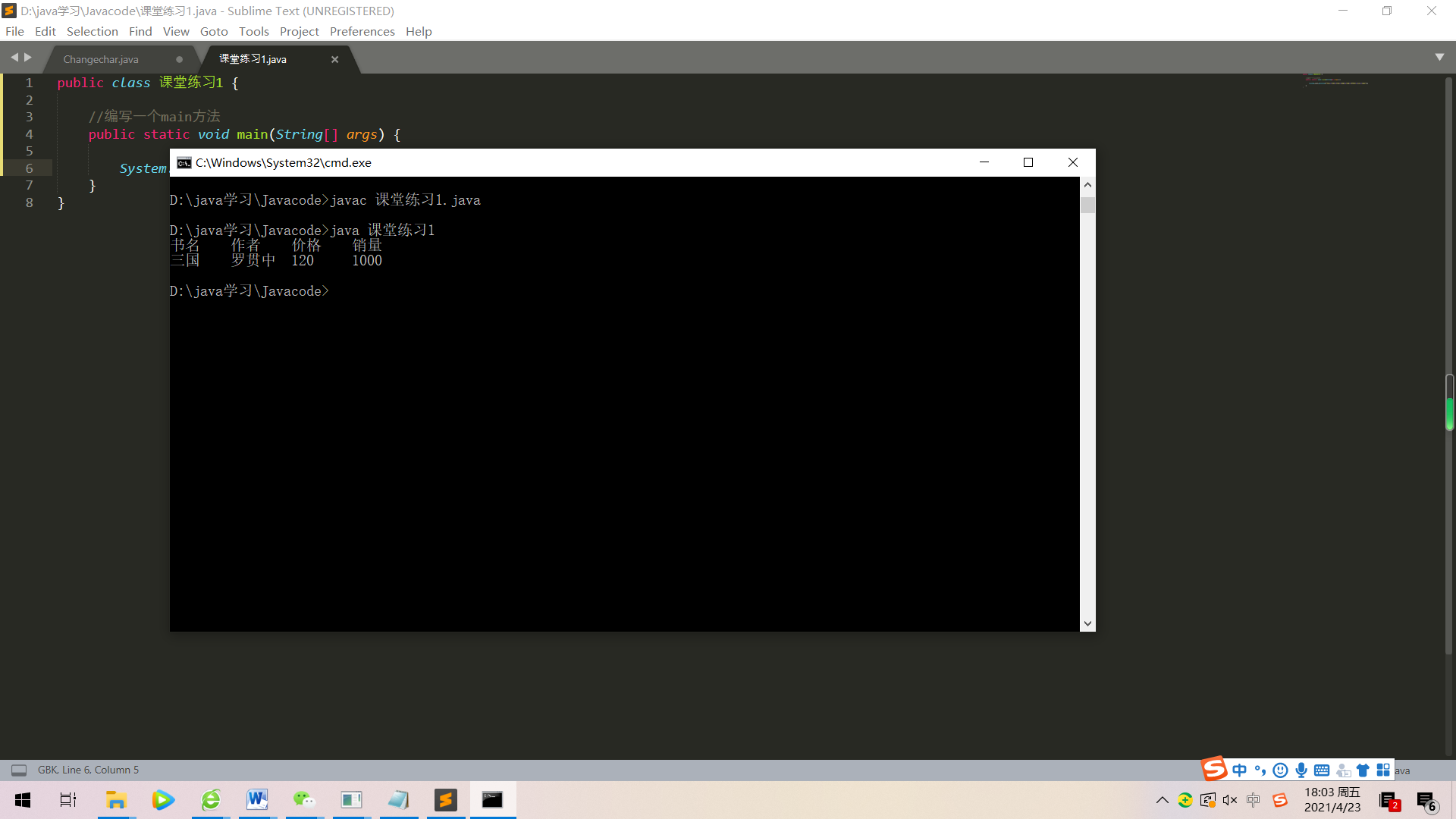
System.out.println("韩顺平教育\r北京");

System.out.println("韩顺平教育\r\n北京"); //换行不会替换

}

}





课堂练习1截图

0021 Java 容易犯的错误

⑴找不到文件：源文件名写错或不存在

⑵文件名和主类名不匹配

⑶缺；，{}，（）根据编译器提示修改即可

⑷找不到方法一般是单词拼写的问题

不好修改的错误是业务错误，环境错误

0022 注释（comment）

用于注释说明解释程序的文字，提高代码的可读性

Java的注释类型：

⑴单行注释

⑵多行注释

⑶文档注释

⑷

0023 单行注释与多行注释

⑴单行注释

格式：//注释内容

⑵多行注释

/\*

注释

注释

\*/

[注]：被注释的文字，不会被JVM即Java虚拟机执行

多行注释里面不允许有多行注释嵌套

0024 文档注释

注释内容可被JDK提供的工具javadoc所解析，生成一套以网页文件形式体现的该程序的说明文档，一般写在类。

Javadoc标签有：

0025 java代码规范

1. 类、方法的注释要以Javadoc的方式来写
2. 非Javadoc的注释着重告诉读者为什么这么写，如何修改，注意什么问题等
3. 使用tab操作，实现整体向右边移动，用shift+tab整体向左移
4. 运算符和=两边习惯各加一个空格
5. 源文件使用UTF-8代码
6. 行宽度不要超过80字符
7. 代码编写行尾风格和次行风格。推荐使用行尾风格

在每一行行尾有{}，再 起始结束都换行输{}

换行输入与之匹配的{}

0026 DOS原理

DOS：磁盘操作系统

md d:\\hsptemp 在d盘创建一个名为hsptemp的目录

rd d:\\hsptemp 删除d盘名为hsptemp的目录

相对路径：从当前目录开始定位，形成的一个路径

绝对路径：从顶级目录开始定位，形成的路径

例子：在test100访问test200:

相对路径：..\..\abc2\test200\helo.txt ..\示上一级目录

绝对路径：d:\ abc2\test200\helo.txt

0028&0029&0030 常用的DOS命令：

1. 查看当前目录有什么内容：dir

例子：dir d:\abc2\test200

1. 切换到其他盘下：盘符号：cd （change directory）

例如D盘切C盘：

Cd/D c:

1. 切换到当前盘的其他目录:有绝对路径，相对路径两种方式
2. 切换到上一级目录：

cd..

1. 切换到根目录：

cd\

1. 查看指定的目录下所有的子级目录

例子：查看d盘的目录

tree：d \

1. 清屏：cls+回车
2. 退出DOS：exit
3. md: 创建目录:

md 文件名1 文件名2

1. rd : 删除目录：

rd 文件名1 文件名2

1. copy：拷贝文件

copy ok.txt e:\ok.txt

将当前路径下的文件拷贝到e盘

1. del：删除文件

del 文件名

1. echo: 输入内容到文件

exho ok > hello.txt

生成名为hello.txt的文本文件，内容为ok

1. type: 可创建空文件

创建空文件：type nul > abc.txt

1. move：剪切

将文件移动到另一个盘：move ok.txt f:\ok.txt

0034 变量原理

变量是程序的基本组成

变量三要素：类型+名称+值

0035 变量概念

概念：变量相当于内存中一个数据存储空间的表示，通过变量名可以访问到变量（值）

变量声明的基本步骤：

⑴声明变量

⑵赋值

⑶输出变量的值：System.out.println(a);

可一步完成： int 变量名=变量值

0036 入门案例

public class var02 {

//编写一个main方法

public static void main(String[] args) {

//记录个人信息

int age = 30;

double score = 90;

char gender = '男';

String name = "king";

System.out.println("人的信息如下");

System.out.println("name");

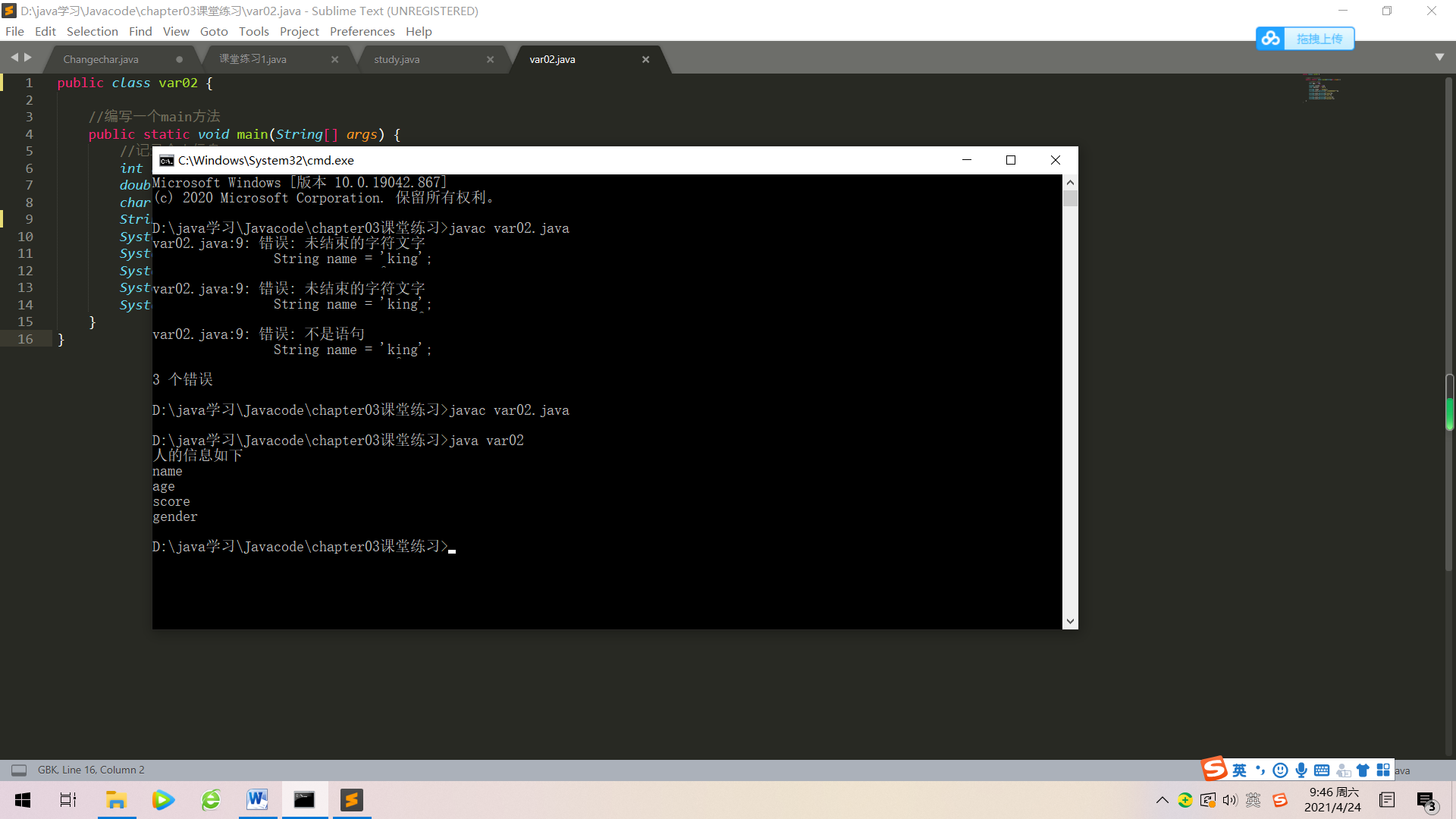
System.out.println("age");

System.out.println("score");

System.out.println("gender");

}

}



0037 变量使用注意事项

1. 变量表示内存中的一个存储区域
2. 该区域有自己的变量名和数据类型
3. 变量必须先声明再使用
4. 该区域的值可在同一类型范围内变化
5. 变量在同一个作用域不能重名
6. 变量三要素：类型+名称+值

0038 +的使用

1. 当左右两边都是数值类型时，做加法运算
2. 当左右两边有一方为字符串，做拼接运算
3. 例子：System.out.println("hello" + 100 + 3);

//hello1003

0039 数据类型

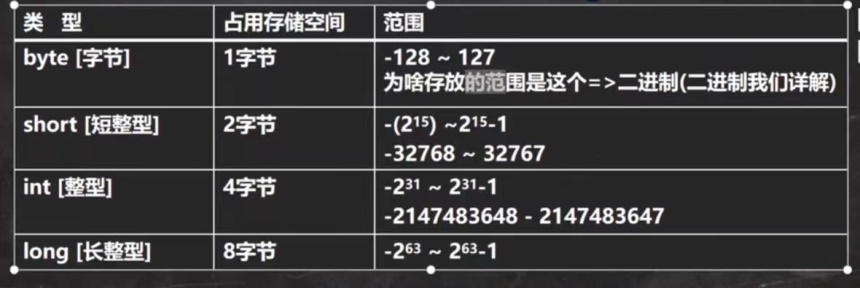
每一种数据都定义了明确的数据类型，在内存中分配了不同大小的内存空间（字节）

Java数据类型

String是一个类

0040 整数类型

Java的整数类型用于存放整数值的



0041 整数类型的使用细节

1. Java各整数类型有固定的范围和字段长度，不受具体OS[操作系统]的影响，以保障Java程序的可移植性
2. Java的整数常量默认为int型，声明long型常量须后加’I’或’L’

注意定义变量时类型的匹配

1. Java程序中变量常声明为int型，除非不足以表示大数，才使用long
2. bit：计算机中的最小存储单位。byte:计算机中基本存储单元，1byte = 8 bit

0042&0043&0044 浮点类型



1. 存放形式：浮点数=符号位+指数位+尾数位
2. 尾数部分可能丢失，造成精度损失(小数都是近似值)

浮点类型使用细节

1. Java各浮点类型有固定的范围和字段长度，不受具体OS[操作系统]的影响，以保障Java程序的可移植性

2. Java的浮点型常量(具体值)默认为double型，声明float型常量，须后加’f’或’F’

3. 浮点型常量有两种表现形式

⑴double num = .123 //等价double num = 0.123

⑵5.12e2 //等价5.12乘十的二次方；5.12e-2 //5.12除十的二次方

4.通常情况下，选用double类型，因为其精度比float高

5.浮点数使用陷阱：当对运算结果是小数的进行相等判断时，要小心

[注]Ctrl+/ 注释快捷键，再次输入就取消注释

public class floatdetail {

//编写一个main方法

public static void main(String[] args) {

double num1 = 2.7;

double num2 = 8.1 / 3;

if(num1 == num2) {

System.out.println("相等");

}

//正确的写法

if(Math.abs(num1-num2) < 0.0000001) {

System.out.println("差值很小，达到规定的精度，认为相等");

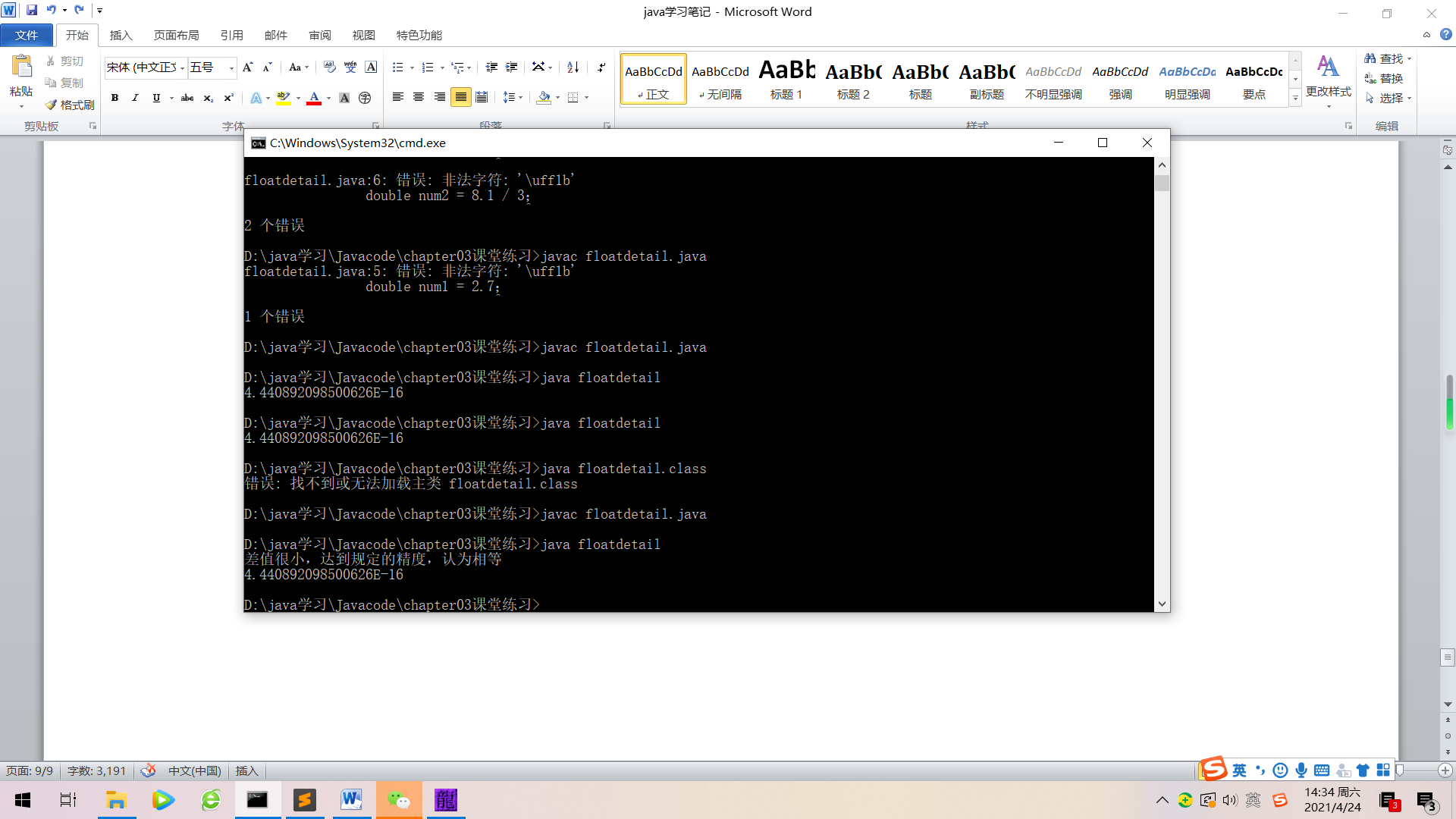
}

//可以通过Java API看

System.out.println(Math.abs(num1-num2));

}

}



0045 Java API文档

1. API:应用程序编程接口，是Java提供的基础编程接口

中文在线文档：<https://www.matools.com>

1. Java语言提供了大量的基础类，有相应的API文档告诉读者类的使用方法以及这些类包含的方法。
2. Java类的组织形式：



1. 举例说明Arrylist类有哪些方法

⑴按：包类方法

⑵直接索引

0046&0047 字符类型（char）

字符串类型可以表示单个字符，字符类型是char，char是两个字节(可以存放汉字)，多个字符用字符串String

字符类型使用细节：

1. 字符用单个’’;” ”表示字符串
2. Java还允许使用转义字符，’\t’和起来表示一个转义字符
3. Java中，char的本质是一个整数，在输出时，是unicode码对应的字符

http://tool.chinaz.com/Tools/Unicode.aspx

1. 可以直接给char赋一个整数，然后输出时，会按照对应的unicode字符整数,要输出整数时，可(int)字符

Char c2 = ‘a’;

System.out.println(int (c2));

1. Char类型是可以进行运算的，相当于一个整数，因为它都有对应的unicode码

0048&0049 字符类型(char)

1. 字符型存储到计算机中，需要将字符对应的码值(整数)找出来
2. 字符和码值对应关系是通过字符码表规定的(规定好的)
3. 字符编码：

⑴ASCLL(ASCLL编码表用一个字节表示，一共有128字符)，缺点是不能表示所有字符

⑵Unicode(Unicode 编码表 固定大小的编码 使用两个字节表示字符，字母和汉字同一都是占用两个字节)，缺点是存储空间浪费。Unicode兼容ASCLL码

⑶Utf-8(编码表，大小可变的编码 字母使用一个字节，汉字使用三个字节)：可使用1-6个字节表示一个符号，根据不同的符号而变化字节长度。

⑷gbk(可以表示汉字，而且范围广泛，字母使用一个字节，汉字使用两个字节)

⑸gbk2312(可以表示汉字，gbk2312<gbk)

⑹big5码(繁体中文，台港，香港)

0050 布尔类型:Boolean

布尔类型数据只允许ture和false

public class bollean {

//编写一个main方法

public static void main(String[] args) {

boolean ispass = false;

if (ispass == true) {

System.out.println("考试通过，恭喜");

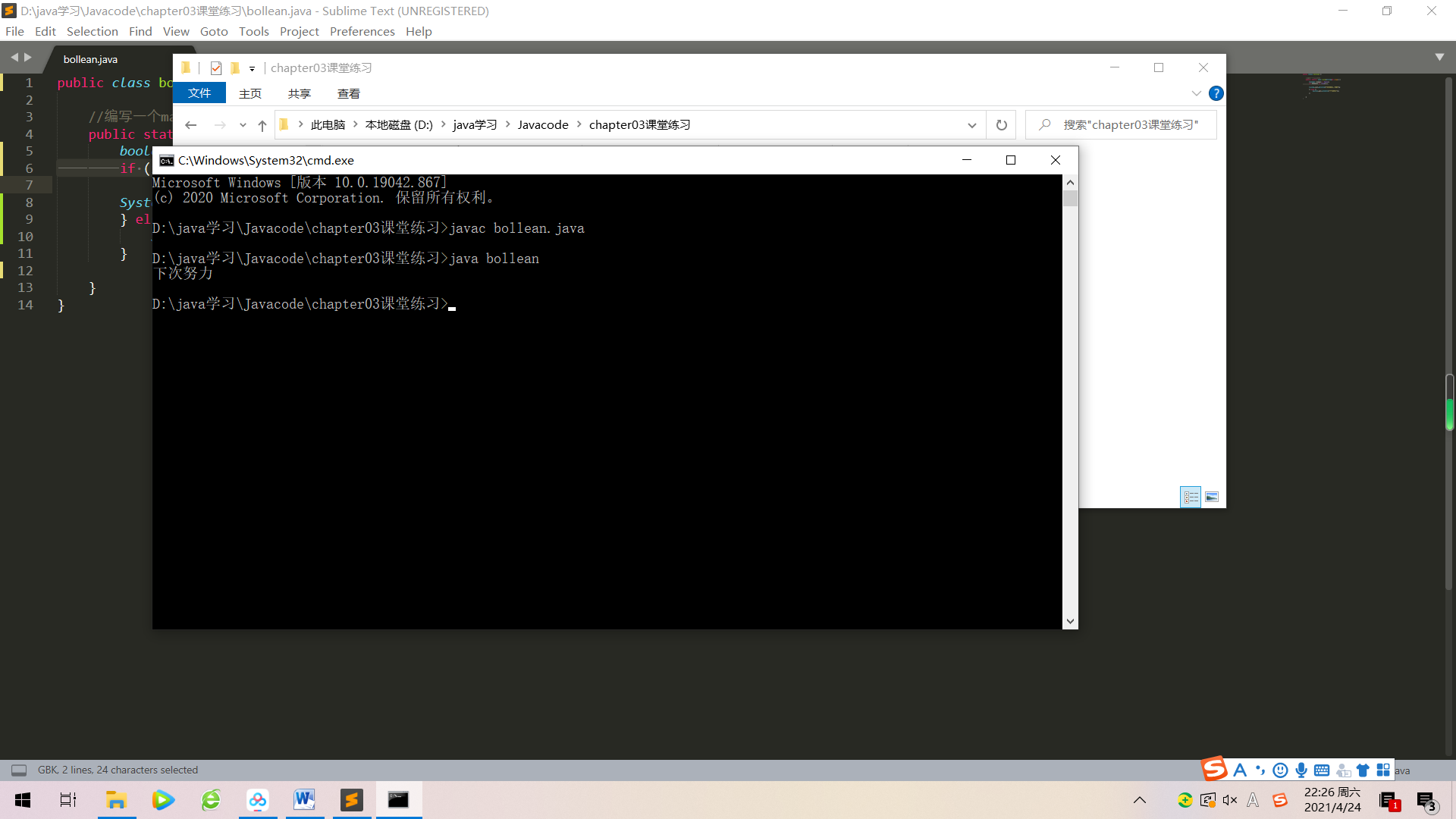
} else {

System.out.println("下次努力");

}

}

}

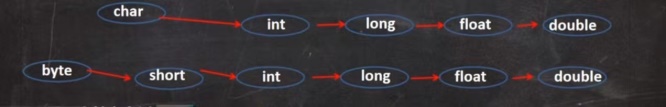


使用细节说明：不可以用0或非0的整数代替true或false

0051&0052&0053&0054&0055&0056 基本数据类型转换

自动类型转换：当java程序在进行时赋值或者运算时，精度小的类型自动转换为精度大的数据类型。

数据类型按精度(容量)大小排序为：



自动类型转换注意和细节

1. 有多种类型数据混合运算时，系统首先自动将所以数据转换成容量最大的那种数据类型，然后再进行计算
2. 当我们把精度大的数据类型赋值给精度小的，数据类型时，就会报错，反之就会进行自动类型转换
3. (byte,short)和char之间不会相互自动转换

当把数赋给byte时，⑴先判断该数是否在byte范围内，如果是就可以

int b1 = 10;//对，-128-127

int n2 = 1;n2是int

byte b2 = n2;//错误，如果是变量赋值，判断类型

byte b = 10

char c1 = b;//错误，b已经是byte,不能自动转换成char

byte c = 1000;//错误，不再-128-127

1. byte,short,char他们三者可以计算，在计算时首先转换为int类型(无论单独运算还是混合运算)

byte b2 = 1;

byte b3 = 2;

int s2 = b2 + b3;//错误：b2 + b3🡺int

1. boolean不参与类型的自动转换
2. 自动提升原则：表达式结果的类型自动提升为操作数中最大的类型

强制类型转换：

自动类型的逆过程，将容量大的数据类型转韩文容量小的数据类型，使用时要加上强制转换符(),但可能造成精度降低或溢出，格外要注意。

public class forceconvert {

//编写一个main方法

public static void main(String[] args) {

int num1 = (int)1.7;

System.out.println("num1=" + num1);//1,造成精度损失

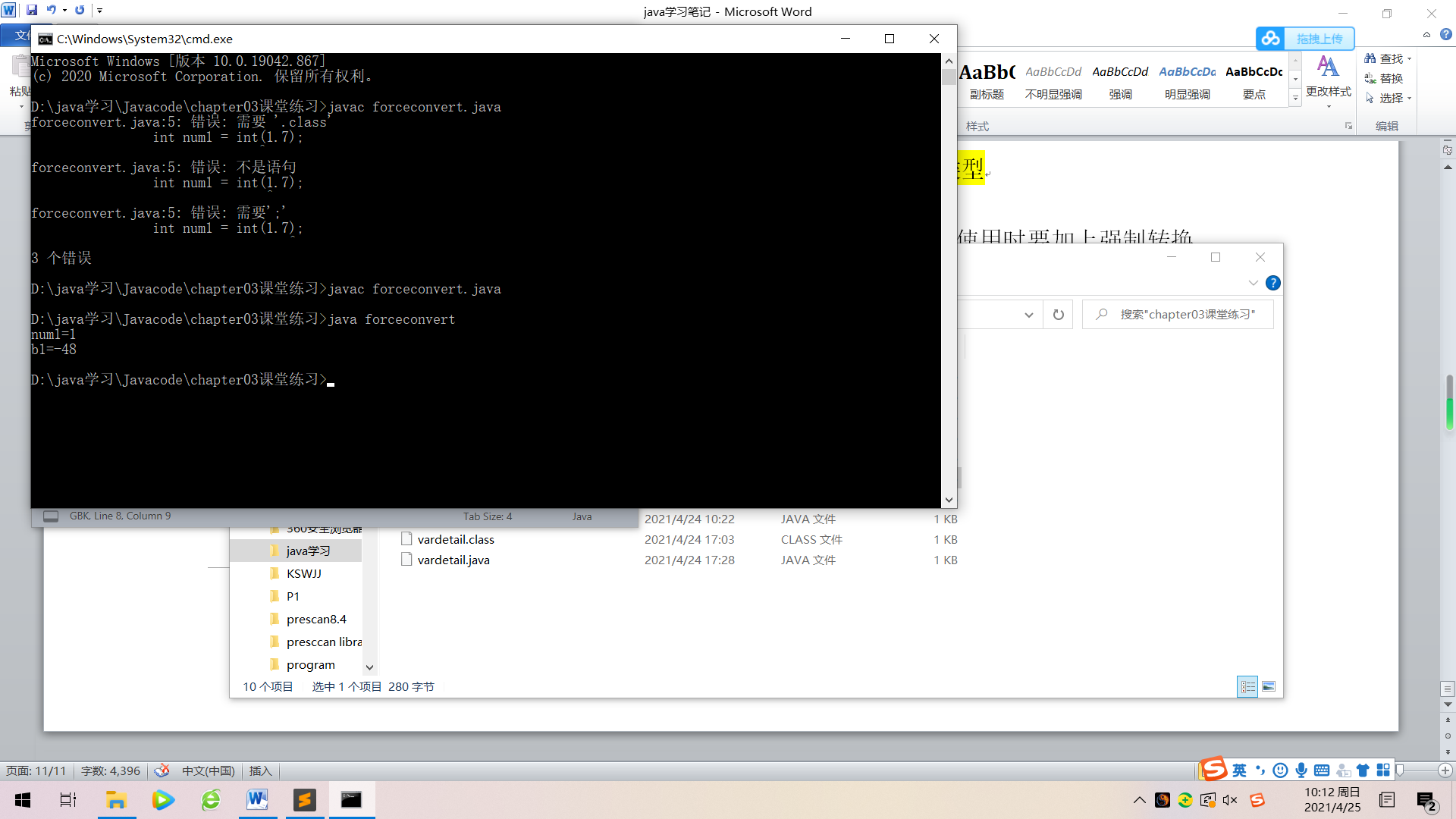
int num2 = 2000;

byte b1 = (byte)num2;

System.out.println("b1=" + b1);//造成数据溢出

}

}



强制类型转换细节

1. 当进行数据的大小从大—>小，需要强制类型转换
2. 强制转换符号支队最近的操作数有效，可使用小括号提升优先级
3. Char类型可以保存int的常量值，但不能保存int 的变量值，需要强制转换

int m = 100;//ok

char c2 = m;//no

char c3 = (char)m;//ok

System.out.println(c3);//100对应的字符

1. byte,short和char类型在进行运算时，当做int类型处理

0057&0058 基本数据类型和String类型的转换

1.基本类型转String类型

语法：将基本类型的值+””即可

public class stringbasic {

//编写一个main方法

public static void main(String[] args) {

int num1 = 100;

float f1 = 1.1f;

double d1 = 4.5;

boolean b1 = true;

String s1 = num1 + "";

String s2 = f1 + "";

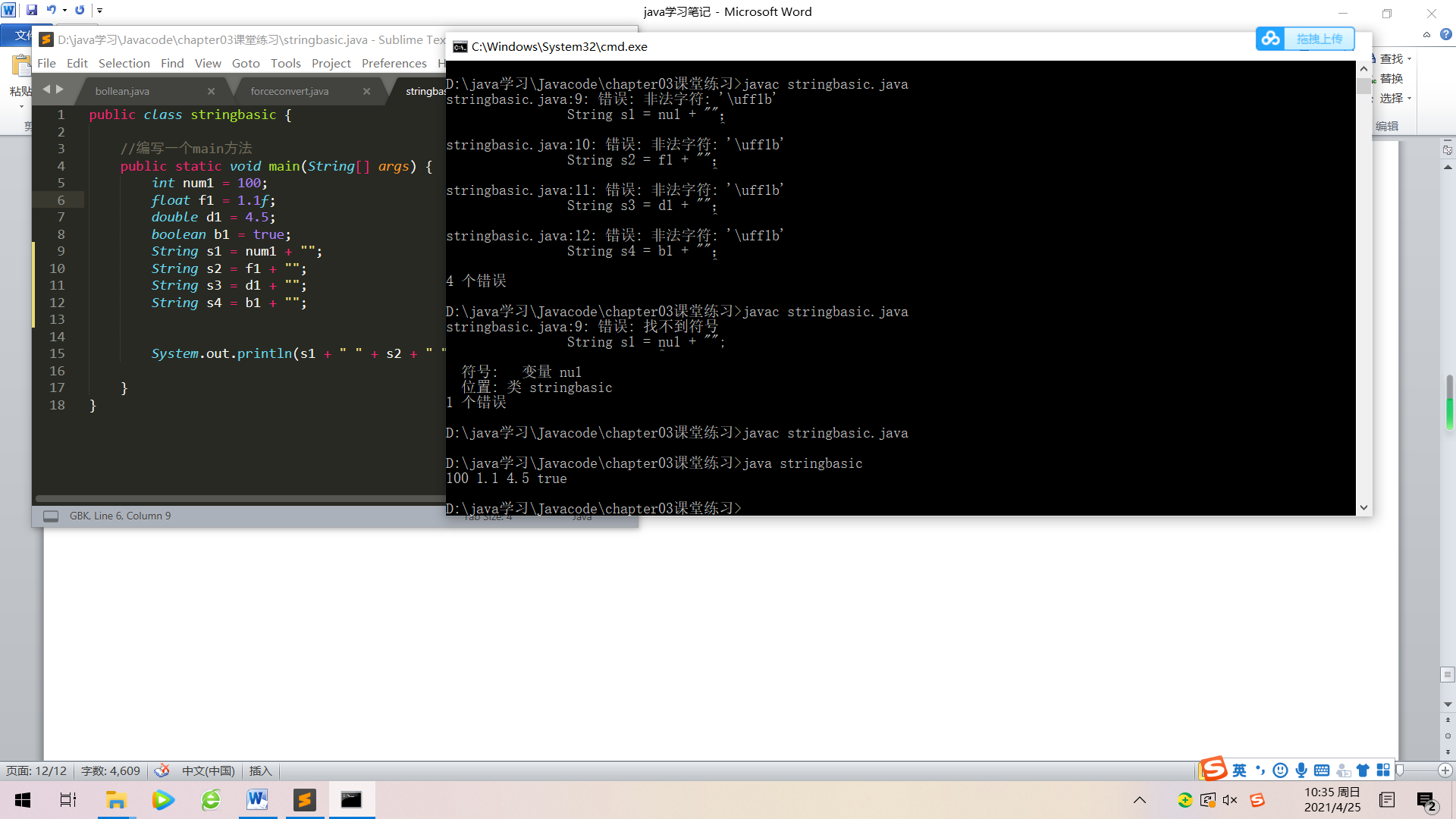
String s3 = d1 + "";

String s4 = b1 + "";

System.out.println(s1 + " " + s2 + " " + s3 + " " + s4 );//1,造成精度损失

}

}



2.String类型转基本数据类型

通过基本类型的包装类调用parseXX方法即可

String s5 = "123";

int n1 = Integer.parseInt(s5);

double n2 = Double.parseDouble(s5);

float n3 = Float.parseFloat(s5);

byte n4 = Byte.parseByte(s5);

long n5 = Long.parseLong(s5);

boolean b = Boolean.parseBoolean("true");

short n6 =Short.parseShort(s5);

System.out.println(n1);

System.out.println(n2);

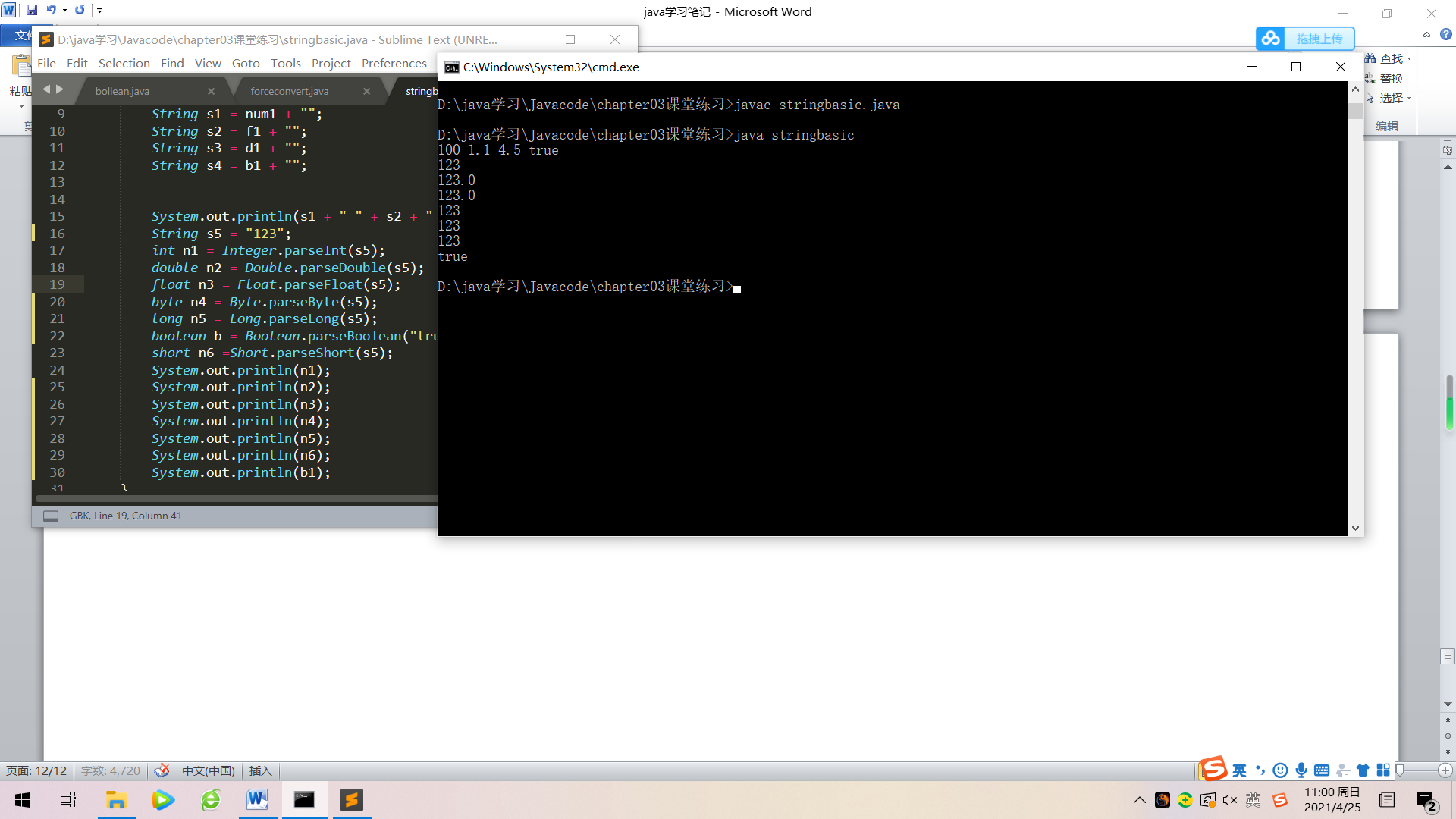
System.out.println(n3);

System.out.println(n4);

System.out.println(n5);

System.out.println(n6);

System.out.println(b1);



[注]：把字符串转换成字符的含义是指把字符串的第一个字符得到

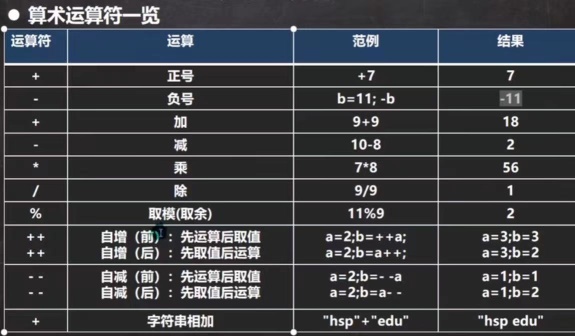
注意事项：

1. 在将String类型转换成基本数据类型时，要确保String类型能转换成有效的数据，比如不能把“hello”转换成一个整数
2. 如果格式不正确，就会抛出异常，程序就会终止

0062 运算符

运算符是一种特殊的符号，用以表示数据的运算、赋值和比较

1. 算数运算符



取模的本质：a % b = a – a / b \* b

自增单独使用两种方式无区别；作为表达式使用，++i先自增再赋值，i++先赋值后自增

public class arithmeticoperator {

//编写一个main方法

public static void main(String[] args) {

System.out.println(10 / 4);//java中是2int

double d = 10 / 4;//2.0double

System.out.println(d);

System.out.println(10.0 / 4);//2.5double

System.out.println(10 % 3);//1

System.out.println(-10 % 3);//-1

System.out.println(10 % -3); //1

System.out.println(-10 % -3);//-1

//+=的使用

int i = 10;

i++;//自增 等价于 i = i+1

++i;//自增 等价于 i = i+1

System.out.println("i=" + i);//-1

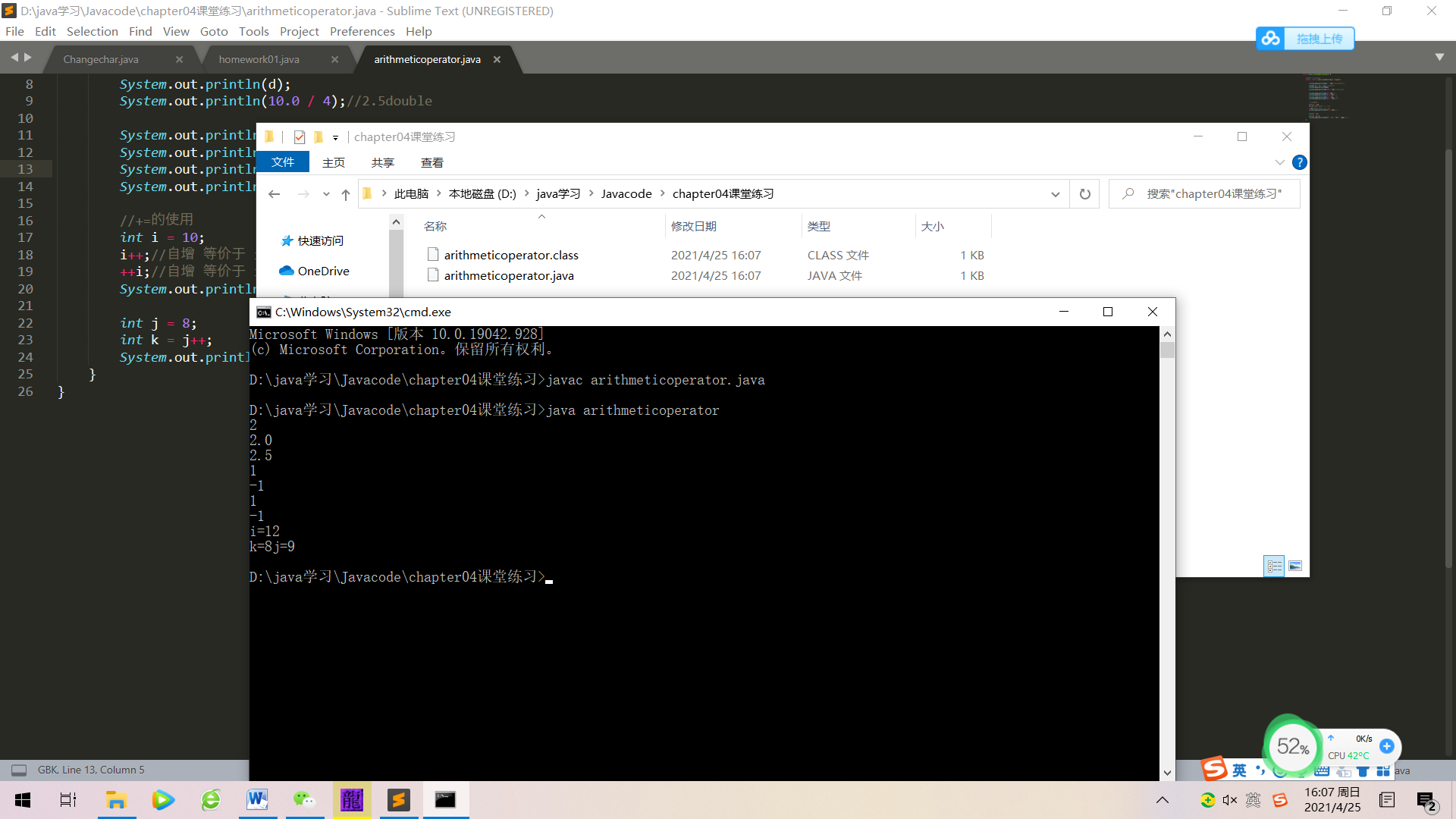
int j = 8;

int k = j++;

System.out.println("k=" + k + "j=" + j);//8 9

}

}



public class arithmeticoperator02 {

//编写一个main方法

public static void main(String[] args) {

int days = 25911;

int weeks = days / 7;

int leftdays = days % 7;

System.out.println(days + "天 合" + weeks +"星期零" + leftdays + "天");

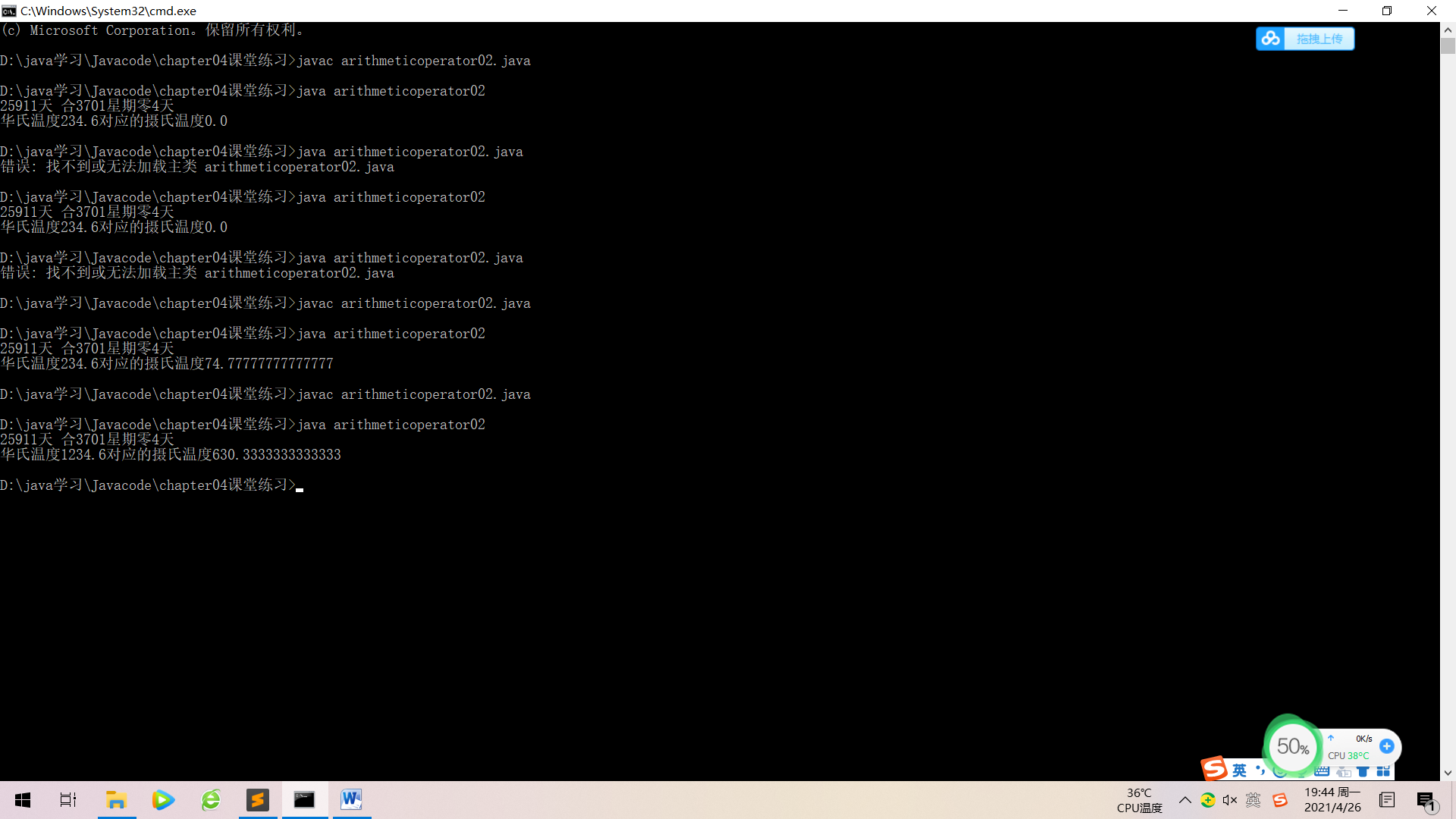
double huashi = 1dr234.6;

double sheshi = 5.0 / 9 \* (huashi - 100);

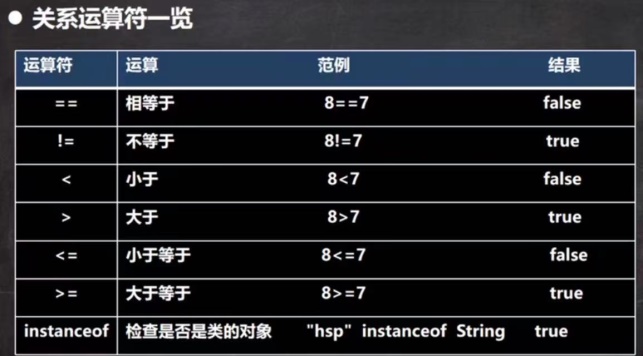
System.out.println("华氏温度" + huashi + "对应的摄氏温度" + sheshi);

}

}



1. 关系运算符(比较运算符)



⑴关系运算符的结果都是boolean型，也就是要么是true，要么是false

⑵关系表达式经常用在if结构的条件中或循环结构的条件中

public class relationaloperator {

//编写一个main方法

public static void main(String[] args) {

int a = 9;

int b = 8;

System.out.println(a > b);

System.out.println(a >= b);

System.out.println(a <= b);

System.out.println(a < b);

System.out.println(a == b);

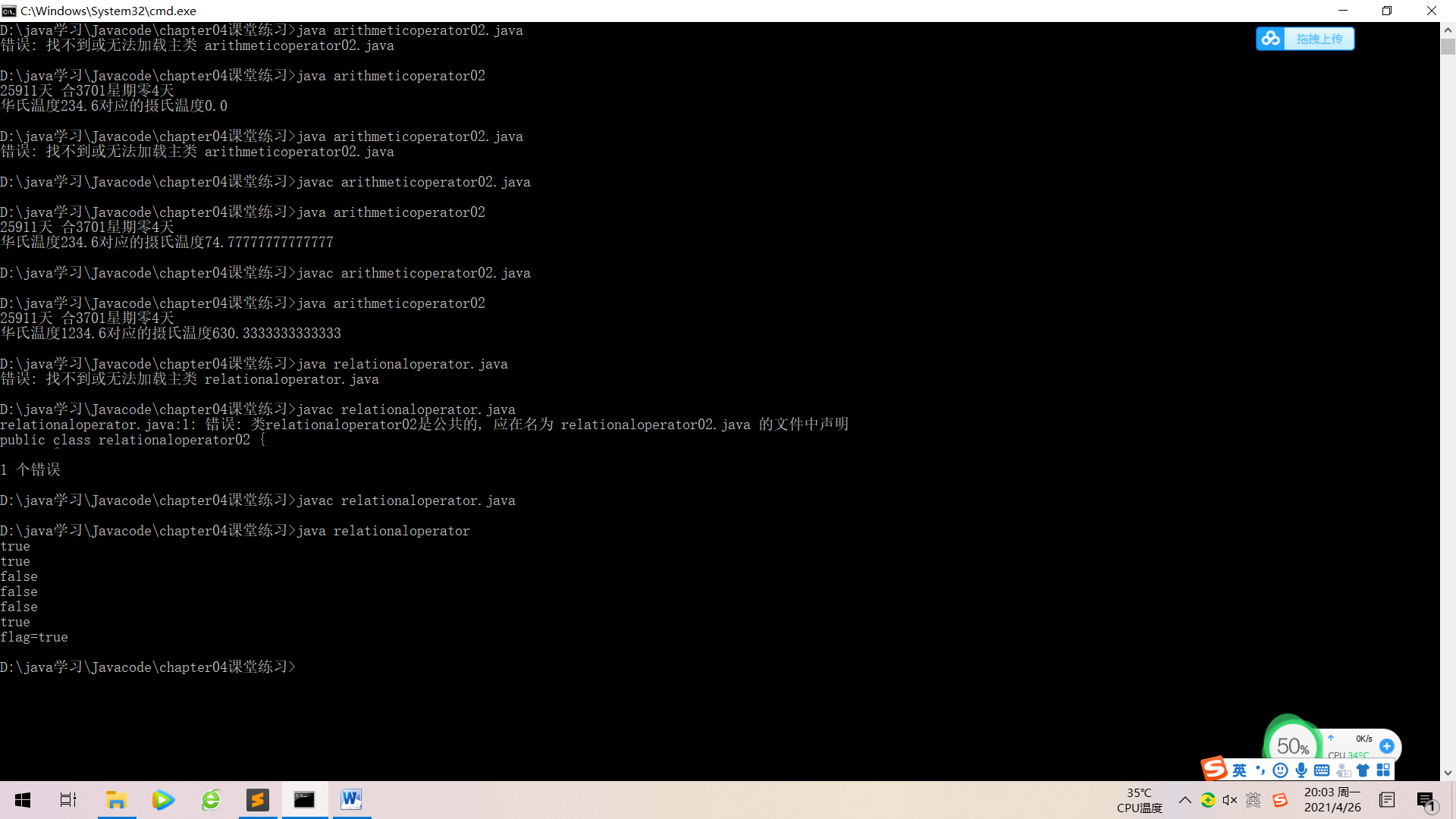
System.out.println(a != b);

boolean flag = a > b;

System.out.println("flag=" + flag);

}

}



1. 逻辑运算符

用于连接多个条件（多个表达式），最终结果也是一个boolean值

⑴短路与 &&，短路或||，取反！

⑵逻辑与&，逻辑或|，^逻辑异或

逻辑运算规则：

1. a&b:&叫逻辑与：规则：当a和b同时为true,则结果为true,否则为false
2. a&&b: &&叫短路与：规则：当a和b同时为true,则结果为true,否则为false
3. a|b:|叫逻辑或：规则：当a和b有一个为 true,则结果为true,否则为false
4. a||b：||叫短路或：规则：当a和b有一个为 true,则结果为true,否则为false
5. ！a:叫取反，或者非运算。规则：当a为 true,则结果为false,否则为true
6. a^b:叫逻辑异或：当a和b不同时,则结果为true,否则为false

0071 短路与逻辑与

区别：

⑴对于&&而言，如果第一个条件为false，后面的条件不再判断

⑵对于&而言，如果第一个条件为false，后面的条件仍然会判断

⑶开发常用&&，效率高

public class logitoperator {

//编写一个main方法

public static void main(String[] args) {

int age = 50;

if(age > 20 && age < 30) {

System.out.println("ok100");

}

if(age > 20 & age < 90) {

System.out.println("ok200");

}

//区别

int a = 4;

int b = 9;

if(a < 1 && ++b < 50) {

System.out.println("ok300");

}

System.out.println("a=" + a + "b" +b);//4 9,对于&，b=10

}

}

0072 ||与|

⑴对于||而言，如果第一个条件为true，后面的条件不再判断,最终结果为true，效率高

⑵对于|而言，如果第一个条件为true，后面的条件仍然会判断

⑶开发常用||，效率高

if(a < 1 || ++b < 50) {

System.out.println("ok300");

}

System.out.println("a=" + a + "b" +b);//4 9,对于|，b=10

0073 ^和！

System.out.println(60 > 20);//T

System.out.println(!(60 > 20));//F

boolean b = (10 > 1) ^ (3 < 5);

System.out.println("b=" + b );//F

0075 赋值运算符

将某个运算后的值，赋给指定的变量。

赋值运算符的种类：

基本赋值运算符：=

复合赋值运算符：+=，-=，/=，%-等

a + =b等价于a = a + b

赋值运算符特点

1. 运算顺序从右往左
2. 赋值运算符的左边只能是变量，右边可以是变量、表达式、常量值
3. 复合运算符等价以下效果：a + =b等价于a = a + b，其他类推
4. 赋值运算符会进行类型转换

byte b = 3;

b+=2;//b=byte(b+2)

b++;//b=byte(b+1)

0077 三元运算符

基本语法：条件表达式？表达式1:表达式2；

运算规则：1.如果条件表达式为true，运算后的结果是表达式1；

2. 如果条件表达式为false，运算后的结果是表达式2；

public class assignoperator {

//编写一个main方法

public static void main(String[] args) {

int n1 = 10;

n1 += 4;

System.out.println(n1);

n1 /=3;

System.out.println(n1);

int a = 10;

int b = 99;

int result = a > b ? a++ : b--;

System.out.println("result=" + result);//99

}

}

使用细节：

1. 表达式1和表达式2要为可以赋给接收变量的类型（或可以自动转换）
2. 三元运算符可以转换成if else的形式

public class ternaryoperator {

//编写一个main方法

public static void main(String[] args) {

int n1 = 553;

int n2 = 33;

int n3 = 123;

int max1 = n1 > n2 ? n1 : n2;

int max2 = max1 > n3 ? max1 :n3;

System.out.println("最大数=" + max2);

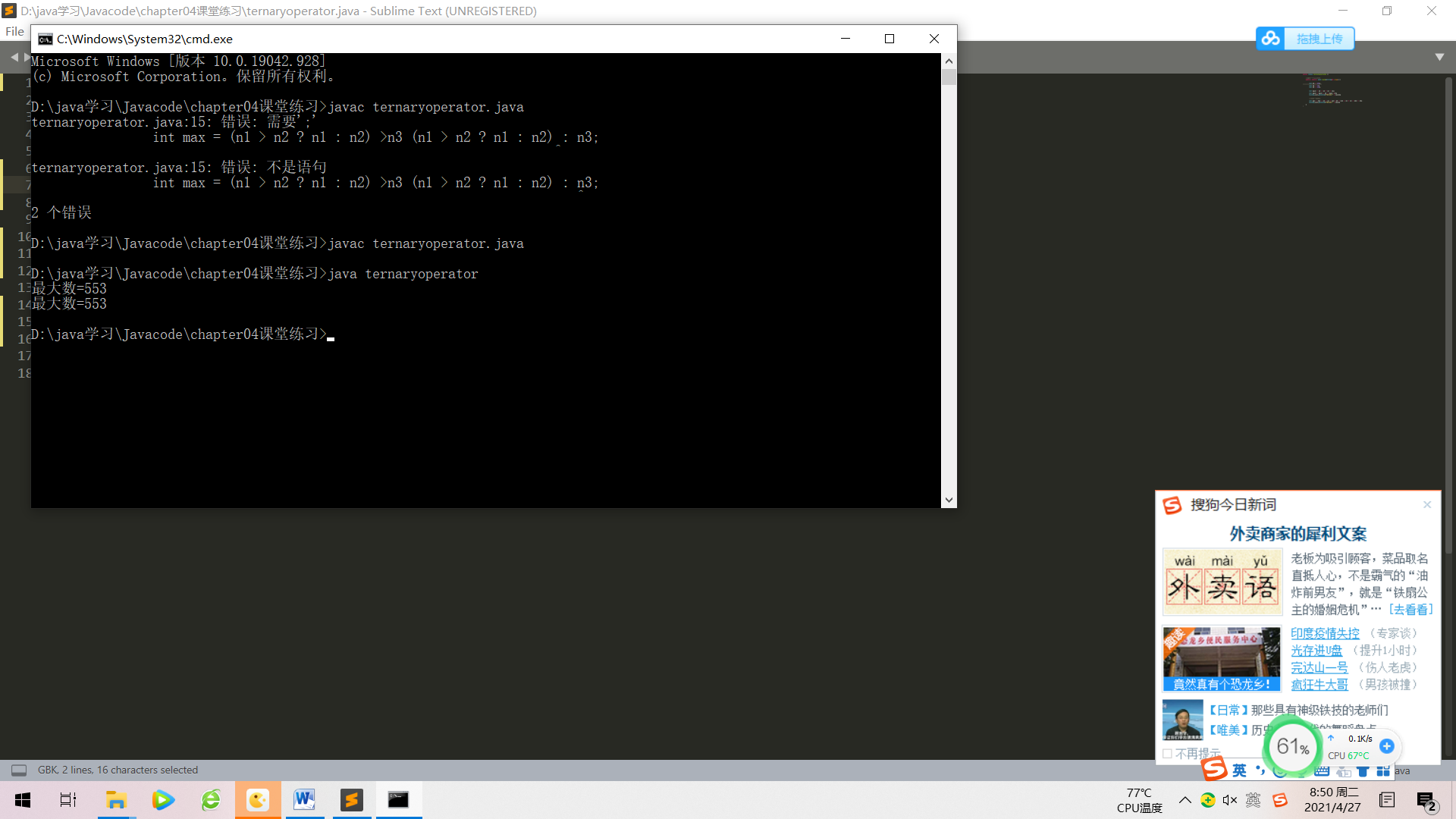
//使用一条语句

int max = (n1 > n2 ? n1 : n2) >n3 ? (n1 > n2 ? n1 : n2) : n3;

System.out.println("最大数=" + max);

}

}



0080 运算符优先级

1. 运算符有不同的优先级，优先级即表达式运算中的运算顺序，如右表，上一行运算符总优先于下一行
2. 只有单目运算符、赋值运算符是从右向左运算的
3. （），{}等
4. 单目运算符++ --
5. 算数运算符
6. 位移运算符
7. 比较运算符
8. 逻辑运算符
9. 三元运算符
10. 赋值运算符

0081 标识符的命名规则(必须遵守)和规范(更专业)

标识符的概念

1. Java对各种变量、方法和类等命名时使用的字符序列称为标识符
2. 凡是可以自己取名字的地方都叫标识符

标识符的命名规则：

1. 由26个英文字母大小写，0-9，\_或$组成
2. 数字不可以开头
3. 不可以使用关键字(已被开发者使用)和保留字(现在未使用，以后会使用)，但可包含关键字
4. Java中严格区分大小写，长度无限制
5. 标识符不能包含空格

标识符命名规范：

1. 包名：由多个单词组成时所有字母都小写：aaa,bbb,ccc
2. 类名、接口名：多个单词组成时，所有单词的首字母大写：XxxYyyZzz
3. 变量名、方法名：多个单词组成时，第一个单词首字母小写，第二个单词开始每个单词首字母大写：xxxYyyZzz
4. 常量名：所有字母都大写。多个单词时每个单词用下划线连接XXX\_YYY\_ZZZ

0082 关键字和保留字

关键字

定义：被JAVA语言赋予了特殊含义，用作专门用途的字符串

特点：关键字中所有字母都为小写

保留字：现有JAVA版本尚未使用，但以后版本可能会作为关键字使用。自己命名标识符要避免使用这些保留字

byValue、cast、future、generic、inner、operator、outer、rest、var、goto、const

0085 键盘输入语句

在编程中，需要接受用户输入的数据，就可以使用键盘输入语句来获取

步骤：

1. 导入该类的所在包
2. 创建该类对象(声明变量)
3. 调用里面的功能

public static void main(String[] args) {

//1.引入/导入Scanner类所在的包

//2.创建Scanner对象，new表示创建一个对象

//scanner就是Scanner类的对象

Scanner scanner = new Scanner(System.in);

//3.接收用户的收入

System.out.println("请输入名字");

String name = scanner.next();//接收用户输入字符串

System.out.println("请输入年龄");

int age = scanner.nextInt();//接收用户输入int

System.out.println("请输入薪水");

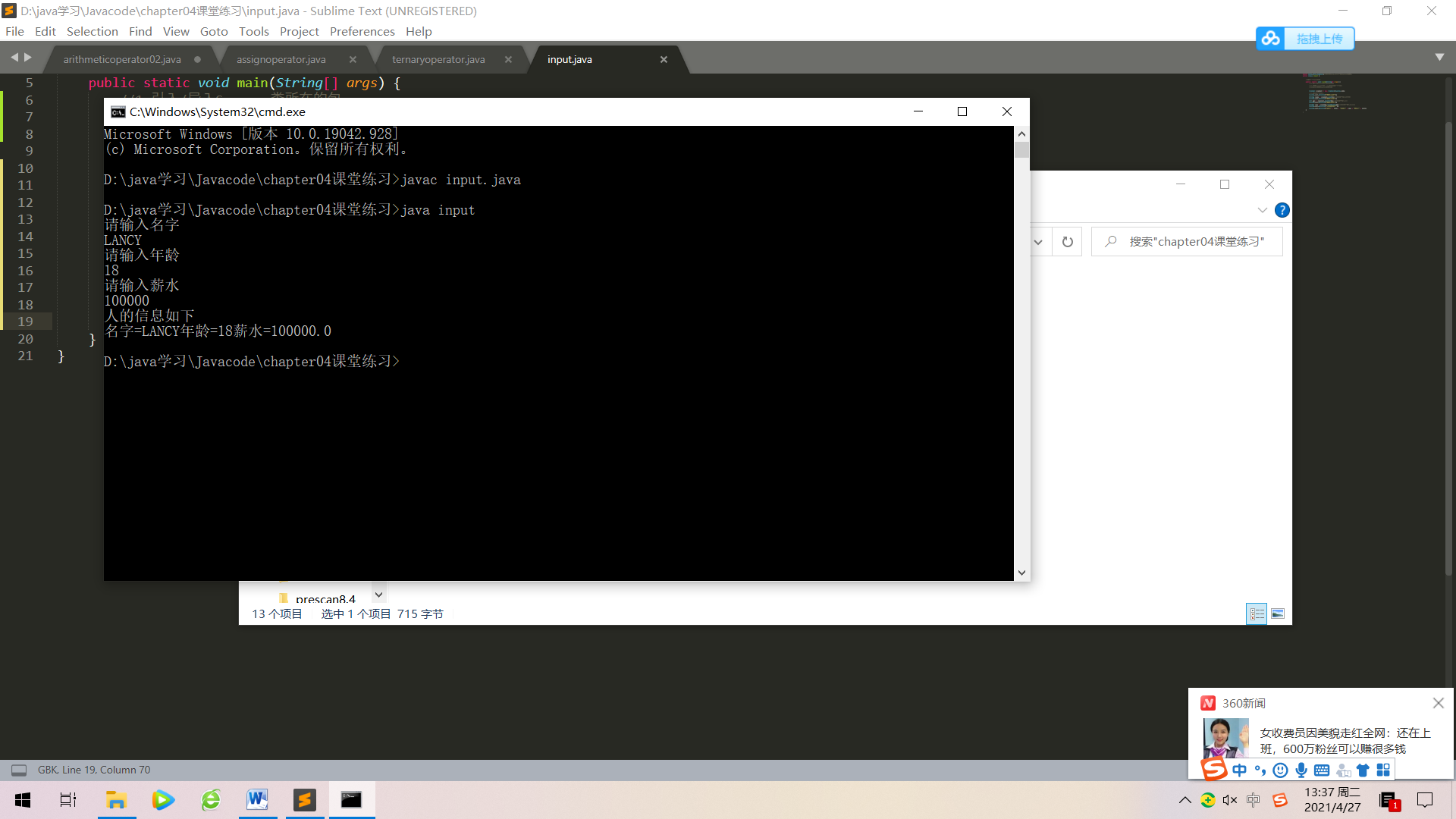
double sal = scanner.nextDouble();//接收用户输入double

System.out.println("人的信息如下");

System.out.println("名字=" + name + "年龄=" + age + "薪水=" + sal);

}

}



0086 进制

对于整数，有四种表示方式：

1. 二进制：0,1，满2进1，以0b或0B开头
2. 十进制:0-9,满10进1
3. 八进制:0-7，满8进1，以数字0开头表示
4. 十六进制：0-9及（10）-F（15），满16进1，以0x或0X开头表示。此处的A-F不区分大小写。

0087-0096 进制的转换

1. n进制转十进制

规则：从最低位（右边）开始，将每个位上的数据提取出来。乘以n的（位数-1）次方，然后求和。

public class binarytest {

//编写一个main方法

public static void main(String[] args) {

int n1 = 10;

int n2 = 1010;

int n3 = 01010;

int n4 = 0x10101;

System.out.println("n1=" + n1);

System.out.println("n2=" + n2);

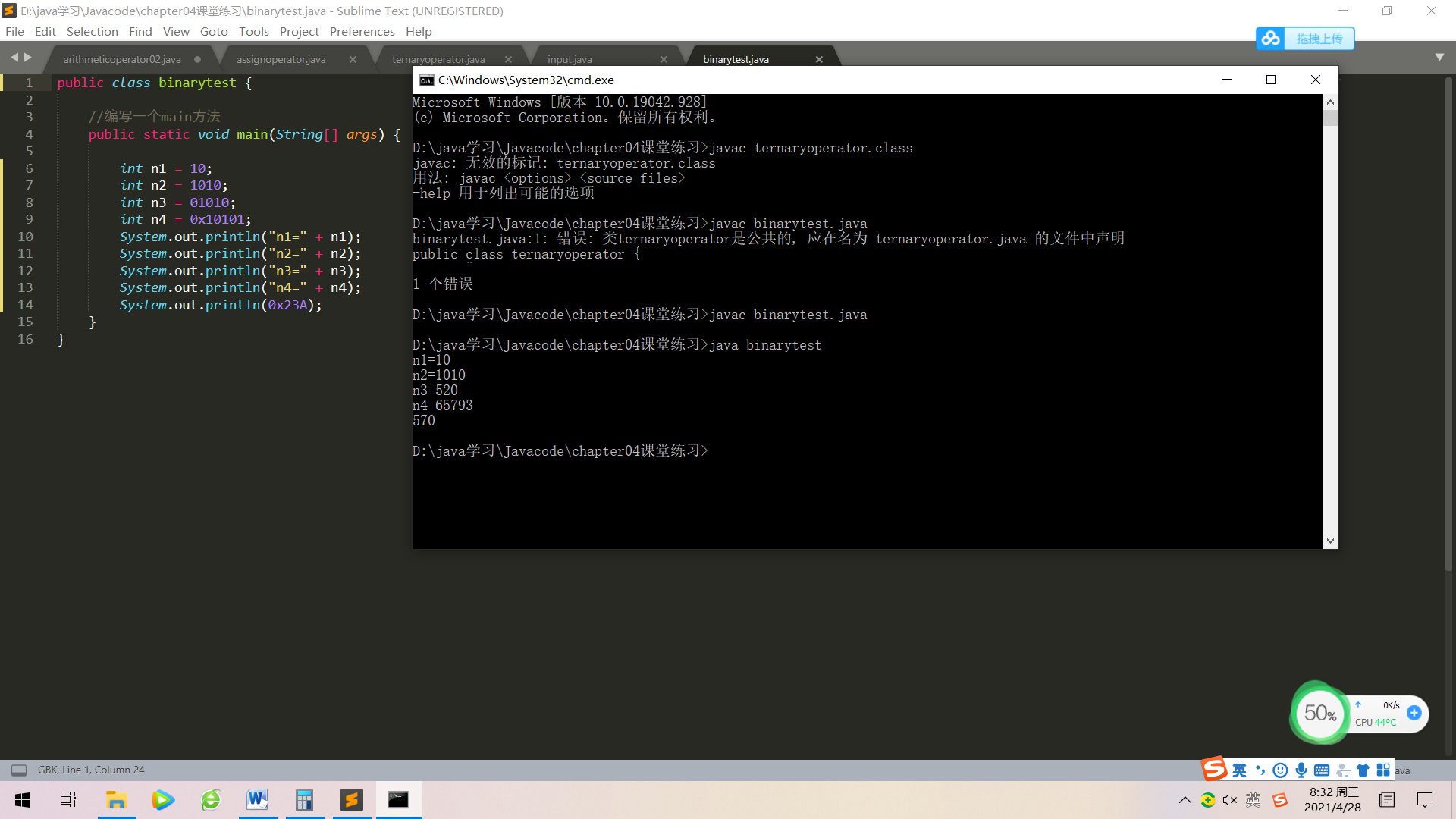
System.out.println("n3=" + n3);

System.out.println("n4=" + n4);

System.out.println(0x23A);

}

}



1. 十进制转其他n进制

规则：将该数不断除以n，直到商为0为止，然后将每步得到的余数倒过来，就是对应的n进制。

1. 二进制转八，十六进制

（1）.转八进制

规则：从低位开始，将二进制数每三位一组，转成对应的八进制数即可

（2）. 转十六进制

规则：从低位开始，将二进制数每四位一组，转成对应的八进制数即可

1. 八进制，十六进制转二进制

（1）.八进制转二进制

规则：将八进制数每1位，转成对应的一个三位的二进制数即可。

（2）.十六进制转二进制

规则：将八进制数每1位，转成对应的一个四位的二进制数即可。

0097 位运算的思考

0098 原码、反码、补码（补码将正数和负数统一起来）

1. 二进制的最高位是符号位；0表示正数，1表示负数
2. 正数的原码、反码、补码都一样
3. 负数的反码=它的原码符号位不变，其他位取反
4. 负数的补码=它的反码+1，负数的反码=它的补码-1
5. 0的反码，补码都是0
6. Java没有无符号数，换言之，java中的数都是有符号的
7. 计算机运行的时候，都是以补码的方式来运算的
8. 当我们看运算结果的时候，要看它的原码（！！）

0099 位运算符

Java中有7个位运算（&、|、^、~、>>、<<和>>>）

按位与&：两位全为1，结果为1，否则为0

按位或|：两位有一个为1，结果为1，否则为0

按位取反：两位有一个为0，一个为1，结果为1，否则为0

public class bitoperator {

public static void main (String[] args) {

// int a = 1>>2;

// int b = -1>>2;

// int c = 1<<2;

// int d = -1<<2;

// int e = 3>>2;

// System.out.println("a=" + a);

// System.out.println("b=" + b);

// System.out.println("c=" + c);

// System.out.println("d=" + d);

// System.out.println("e=" + e);

System.out.println(2&3);

//1.-2的原码10000000 00000000 00000000 00000010

//2.-2的反码11111111 11111111 11111111 11111101

//3.-2的补码11111111 11111111 11111111 11111110

//4.~-2操作 00000000 00000000 00000000 00000001//运算后的补码为正

System.out.println(~-2);//1

//1.-2的原码00000000 00000000 00000000 00000010

//2.-2的反码00000000 00000000 00000000 00000010

//3.-2的补码00000000 00000000 00000000 00000010

//4.~-2操作 11111111 11111111 11111111 11111101//运算后的补码为负，要推回原码

//5.运算后的反码 11111111 11111111 11111111 11111100

//6.运算后的原码 10000000 00000000 00000000 00000011

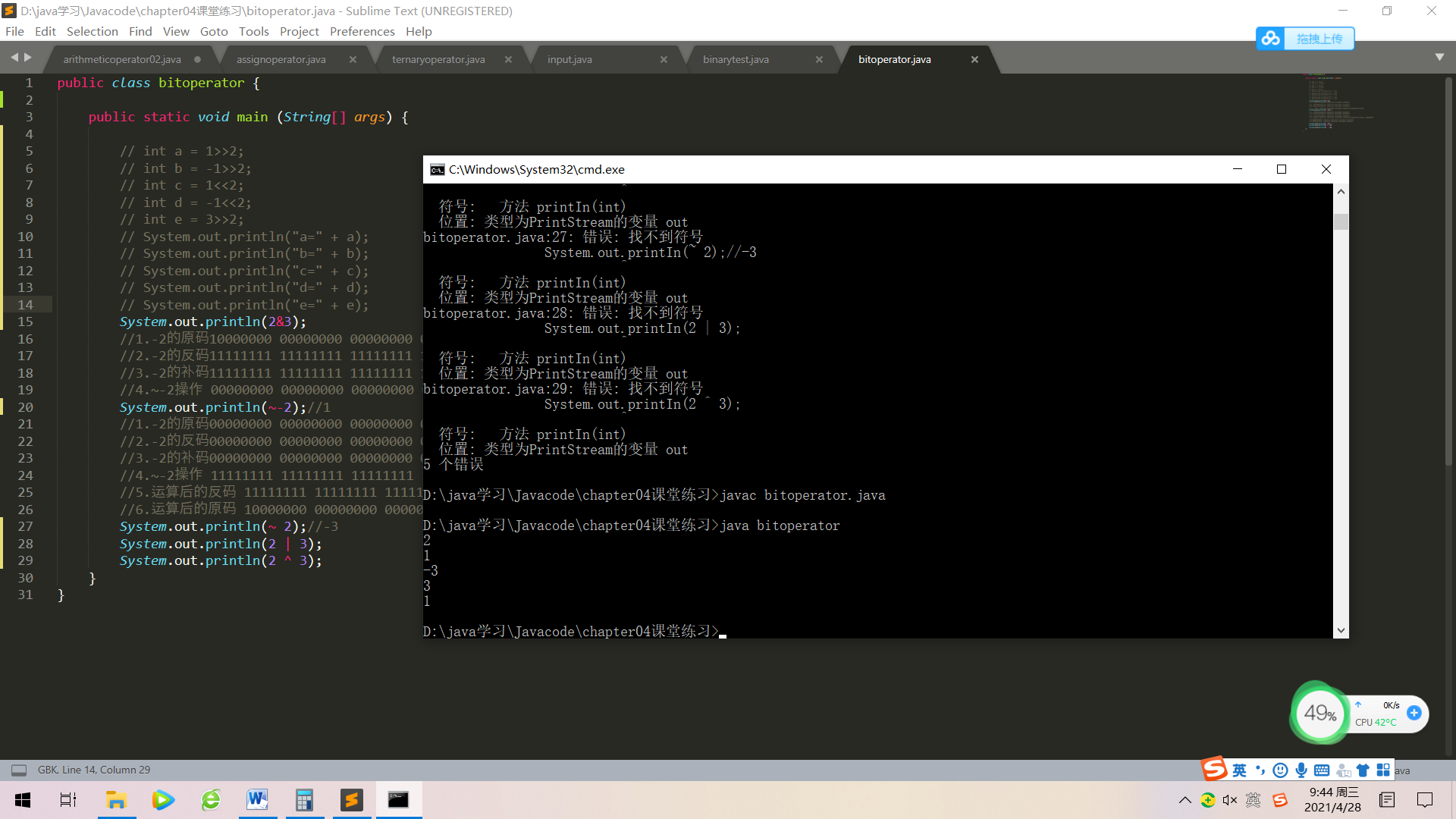
System.out.println(~ 2);//-3

System.out.println(2 | 3);

System.out.println(2 ^ 3);

}

}



0100 位运算符

还有三个位运算符>>> 、<<和 >>，运算规则：

1. 算数右移>>:低位溢出，符号位不变，并用符号位补溢出的高位（本质是移一位，除一次2）
2. 算数左移<<:符号位不变，低位补0（本质是移一位，乘一次2）
3. >>>逻辑右移也叫无符号位右移，运算规则是：低位溢出，高位补0
4. 特别说明：没有<<<符号

0103&0104&0105&0106&0107&0108&0109 程序流程控制介绍

在程序中，程序运行的流程控制决定程序是如何执行的，主要有三大流程控制语句。

1. 顺序控制:程序从上到下逐行执行，中间没有任何判断和跳转

注意java中定义变量时采用合法的前向引用

1. 分支控制：让程序有选择的执行，分支控制有三种
2. 单分支：

基本语法：

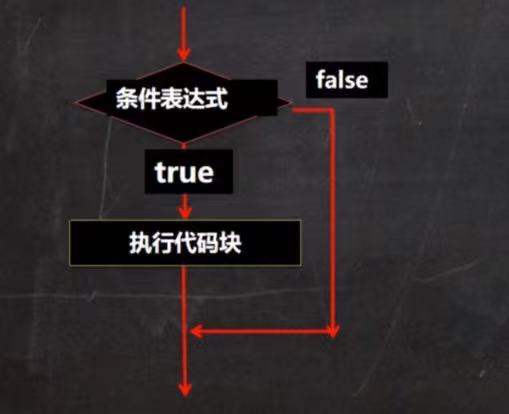
if(条件表达式){

执行代码块；(可以有多条语句.)

}

说明：当条件表达式为true时：，就会执行{}的代码。如果为false，就不执行，特别说明，如果{}中只有一条语句，则可以不用{}，但建议写上。

单分支对应的流程图：



1. 分支控制if-else

基本语法:

If(条件表达式){

执行代码块1；

}

else{

执行代码块2；

}

说明：当条件表达式成立，即执行代码块1，否则执行代码块2。如果执行代码块只有一条语句，则{}可以省略，否则，不能省略。

import java.util.Scanner;

public class if02{

public static void main(String[] args) {

Scanner myscanner = new Scanner(System.in);

System.out.println("请输入你的年龄");

int age = myscanner.nextInt();

if (age > 18) {

System.out.println("你年龄大于18，要对自己的行为负责");

} else {

System.out.println("你的年龄不大");

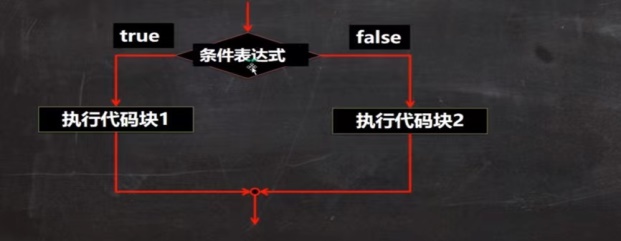
}

System.out.println("程序继续");

}

}

双分支流程图：



public class if03{

public static void main(String[] args) {

int x = 7;

int y = 4;

if (x > y) {//t

if(y > 5) {//f

System.out.println(x+y);

}

System.out.println("韩顺平教育");

} else

System.out.println(" x is" + x);

double num1 = 19.0;

double num2 = 8.0;

double num3 = num1+num2;

if(num1 > 10.0 && num2 <20.0) {

System.out.println(num3);

}

int a = 10;

int b = 12;

int c = a+b;

if(c % 3 == 0 && c % 5 == 0) {

System.out.println("可以整除");

} else {

System.out.println("不能整除");

}

int year = 1000;

if( (year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {

System.out.println("是闰年");

} else {

System.out.println("不是闰年");

}

}

}

1. 多分支

基本语法：

if(条件表达式1) {

执行代码块1；

}

else if(条件表达式2) {

执行代码块2；

}

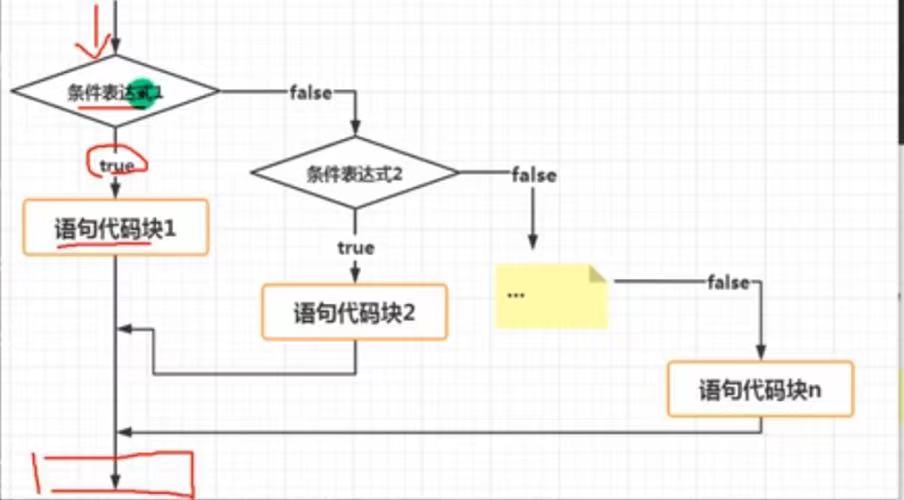
else{

执行代码块n;

}

[注]：多分支可以没有else，如果所有的条件表达式都不成立，则一个执行入口都没有；如果有else，所有的条件表达式都不成立，则默认执行else代码块。

多分支流程图：



说明：

1. 当条件表达式1成立时，即执行代码块1
2. 如果表达式1不成立，才去判断执行代码块2是否成立
3. 如果表达式2成立，就执行代码块2
4. 以此类推，如果所有表达式都不成立，则执行else代码块，注意，只有一个执行入口

import java.util.Scanner;

public class if04{

public static void main(String[] args) {

Scanner myscanner = new Scanner(System.in);

System.out.println("请输入信用分");

int grade = myscanner.nextInt();

//先对输入的信用分进行一个范围的有效判断1-100，否则提示输入错误

if(grade >=1 && grade <= 100) {

if(grade == 100) {

System.out.println("信用极好");

} else if (grade >80 && grade <=99) {

System.out.println("信用优秀");

} else if (grade >= 60 && grade <=80) {

System.out.println("信用一般");

} else {

System.out.println("信用不及格");

}

} else {

System.out.println("信用分需要在1-100");

}

}

}

0112 嵌套分支

在一个分支结构中又完整的嵌套了另一个完整的分支结构，里面的分支的结构称为内层分支，外面的分支结构称为外层分支。

import java.util.Scanner;

public class if05{

public static void main(String[] args) {

Scanner myscanner = new Scanner(System.in);

System.out.println("请输入该歌手的成绩");

double score = myscanner.nextDouble();

if( score > 8.0) {

System.out.println("请输入性别");

char gender = myscanner.next().charAt(0);//把字符串转换成字符

if (gender == '男') {

System.out.println("进入男子组");

} else if(gender == '女') {

System.out.println("进入女子组");

} else {

System.out.println("重新输入性别");

}

} else {System.out.println("淘汰");

}

}

}

0114&0115&0116&0117 switch分支结构

基本语法：

Switch(表达式) {

case 常量1：//当

语句块1；

break；

case 常量1：//当

语句块1；

break；

case 常量n：//当

语句块n；

break；

default;

defalut语句块；

break

}

解读switch：

1. switch关键字，表示switch分支
2. 表达式对应一个值
3. case常量1：当表达式的值等于常量1，就执行语句块1
4. break：表示退出switch
5. 如果和case常量1匹配，就执行语句块1，如果没有匹配，就继续匹配case常量2
6. 如果一个都没有匹配上，就执行default

Switch流程图：

import java.util.Scanner;

public class switch01{

public static void main(String[] args) {

Scanner myscanner = new Scanner(System.in);

System.out.println("请输入一个字符a-g");

char c1 = myscanner.next().charAt(0);

switch(c1) {

case 'a':

System.out.println("一");

break;

case 'b':

System.out.println("二");

break;

case 'c':

System.out.println("三");

break;

case 'd':

System.out.println("四");

break;

case 'e':

System.out.println("五");

break;

case 'f':

System.out.println("六");

break;

case 'g':

System.out.println("日");

break;

default:

System.out.println("你输入的字符不正确，没有匹配的");

}

System.out.println("退出了switch,继续执行程序");

}

}

switch 分支结构

1. 表达式数据类型，应和case后的常量类型一致，或者是可以自动转成可以相互比较的类型，比如输入的是字符，而常量是int
2. switch表达式返回的必须是：（byte,short,int,char,enum[枚举],String）
3. case子句中的值必须是常量（1，’a’）或者是常量表达式,而不能是变量（计算得到的常量也可以）
4. default子句是可选的，当没有匹配的case时，执行default；如果没有default子句，又没有匹配的case时，则没有输出
5. break语句用来在执行完一个case分支后使程序跳出switch语句块；如果没有写break，程序会顺序执行到switch结尾，除非执行到break

import java.util.Scanner;

public class switch03{

public static void main(String[] args) {

Scanner myscanner = new Scanner(System.in);

System.out.println("请输入a-e");

char c1 = myscanner.next().charAt(0);

switch(c1) {

case 'a':

System.out.println("A");

break;

case 'b'://写成'2'也可以，'2'也为字符类型

System.out.println("B");

break;

case 'c':

System.out.println("C");

break;

case 'd':

System.out.println("D");

break;

case 'e':

System.out.println("E");

break;

default:

System.out.println("请重新输入");

}

System.out.println("请输入学生成绩");

double score = myscanner.nextDouble();

int a = (int)(score/60);

if (score >= 0 && score <=100) {

switch(a) {

case 0:

System.out.println("不及格");

break;

case 1:

System.out.println("及格");

break;

default:

System.out.println("成绩要在0-100");

}

} else {

System.out.println("请重新输入");

}

System.out.println("请输入月份");

int month = myscanner.nextInt();

switch(month) {

case 3:

case 4:

case 5:

System.out.println("这是春季");

break;

case 6:

case 7:

case 8:

System.out.println("这是夏季");

break;

case 9:

case 10:

case 11:

System.out.println("这是秋季");

break;

case 12:

case 1:

case 2:

System.out.println("这是冬季");

break;

default:

System.out.println("请重新输入");

}

}

}

0120 switch和if的比较

1. 如果判断的具体数值不多，而且符合byte,short,int,char,enum[枚举],String这六种类型，建议使用switch语句
2. 其他情况，对区间判断，对结果为boolean类型判断，使用if，if的使用范围更广

0121 for循环控制

基本语法：

for(循环变量初始化；循环条件；循环变量迭代) {

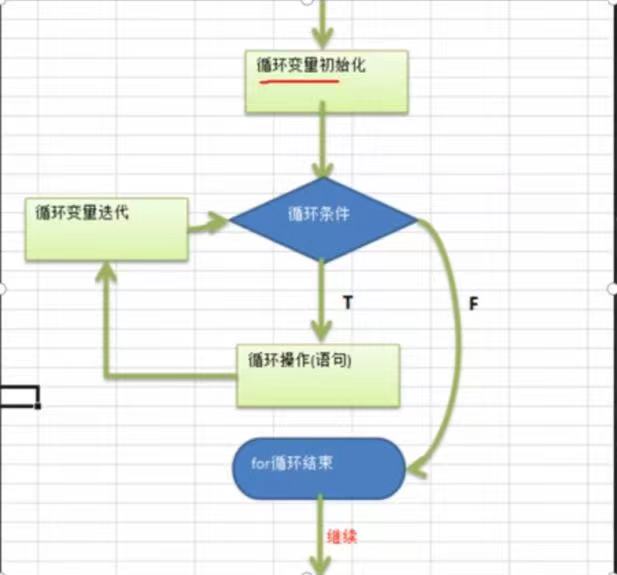
循环操作(可以多条语句)；

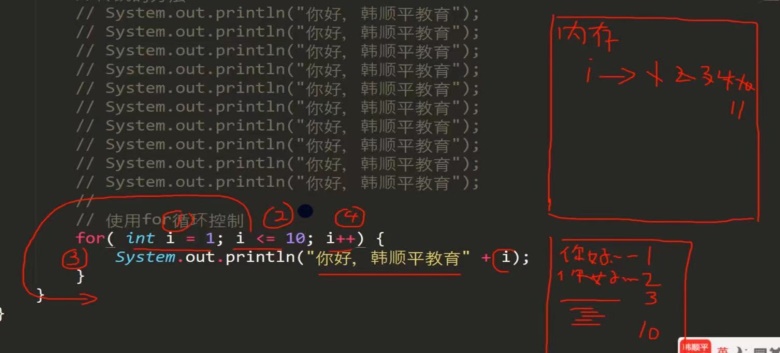
}

说明：

1. for关键字，表示循环控制
2. for有四要素：
3. 循环变量初始化；
4. 循环条件；
5. 循环操作
6. 循环变量迭代
7. 循环操作，这里可以有多条语句，也就是我们要循环执行的代码
8. 如果循环操作（语句）只有一条，可以省略{}，建议不要省略

for循环执行流程





for循环注意事项和细节说明

1. 循环条件是返回一个布尔值的表达式
2. for(；循环判断条件；)中的初始化和变量迭代可以写到其他地方，但是两边的分号不能省略
3. 循环初始值可以有多条初始化语句，但要求类型一样，并且中间用逗号隔开，循环变量迭代也可以有多条变量迭代语句，中间用逗号隔开

public class for02{

public static void main(String[] args) {

int i = 1;

int sum = 0;

int count = 0;

int start = 1;

int end = 100;

int t = 9;

for (i=start;i<=end;i++) {

if (i % t ==0) {

sum += i;

count +=1;

}

}

System.out.println(sum);

System.out.println(count);

//化繁为简，先死后活

int m = 0;

for(;m<=5;m++) {

System.out.println(m + "+" + (5-m) +"=5");

}

}

}

0126 while基本语法

循环变量初始化；

While（循环条件）{

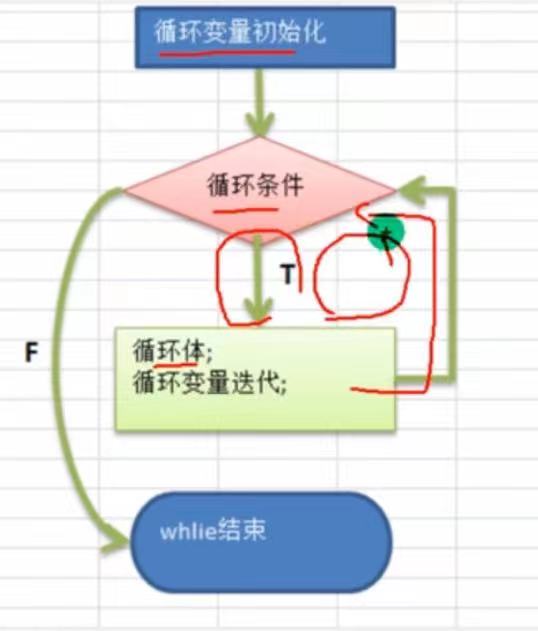
循环体（语句）；

循环变量迭代；

}

while循环也有四要素，只是放置的位置不一样

while循环执行流程



public class while01{

public static void main(String[] args) {

int i = 1;

while(i<=10) {

System.out.println("你好，韩顺平教育" + i);

i++;

}

System.out.println("退出while，继续");

}

}

while注意事项和细节说明

1. 循环条件是返回一个布尔值的表达式
2. while循环是先判断再执行语句

0129 do..while 循环控制

基本语法：

循环变量初始化：

do{

循环体(语句)；

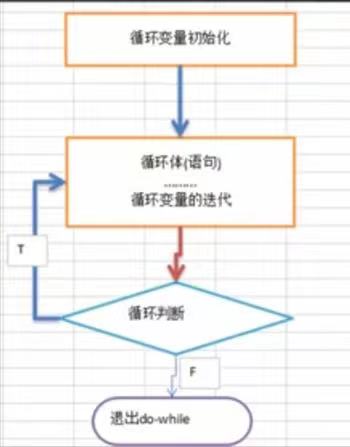
循环变量迭代；

}while(循环条件)；

说明：

1. do while 是关键字
2. while循环也有四要素，只是放置的位置不一样
3. 先执行，再判断，也就是说，一定会至少执行一次
4. 最后有一个分号；
5. while和do while区别：

do while循环执行流程分析：



import java.util.Scanner;

public class dowhile02{

public static void main(String[] args) {

Scanner myscanner = new Scanner(System.in);

char answer = ' ';

do {

System.out.println("还钱吗");

answer = myscanner.next().charAt(0);

System.out.println("他的回答是" + answer);

}while( answer != 'y');

System.out.println("退出while，继续");

}

}

0133 重循环控制

1. 将一个循环多放在另一个循环体内，就形成了嵌套循环，其中，for,while,do…while均可以作为外层循环和内层循环。[建议一般使用两层，最多不要超过3层，否则，代码的可读性很差]
2. 实质上，嵌套循环就是把内层循环当成外层循环的循环体。当只有内层循环的循环条件为false时，才会完全跳出内层循环，才可结束外层的当次循环，开始下一次的循环
3. 设外层循环次数为m次，内层为n次，则内层循环体实际上需要执行m\*n次。

0137 跳转控制语句-break

基本介绍：

break语句用于终止某个语句块的执行，一般使用在switch或者循环（for,while,do-while）中

基本语法：

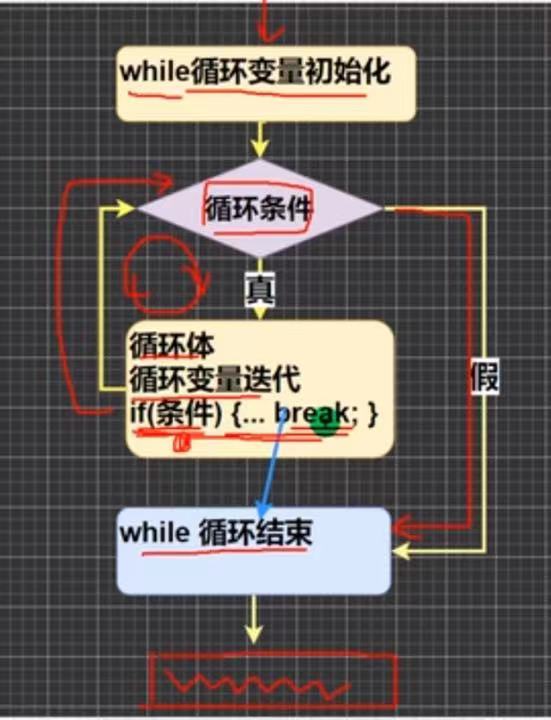
{ ……

Break;

……

}

执行流程：



import java.util.Scanner;

public class break01{

public static void main(String[] args) {

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

if (i == 3) {

break;

}

System.out.println("i=" + i);

}

}

}

注意事项和细节说明：

1. Break 语句出现在多层嵌套的语句块中时，可以通过标签指明要终止的是哪一层语句块
2. 标签的基本使用

lable 1:( …….

lable 2: {…..

lable3: { ……..

break lable2;

……

}

}

}

说明：

1. break语句可以指定退出那层
2. lable1是标签，名字由程序员指定
3. break后指定到那个lable就退出那一层循环
4. 在实际的开发中，尽量不要使用标签
5. 如果没有指定break，默认退出最近的循环体

import java.util.Scanner;

public class break002{

public static void main(String[] args) {

Scanner myscanner = new Scanner(System.in);

int chance = 3;

String name = "";

String passwd = "";

for(int i = 1;i <= 3;i++) {

System.out.println("请输入姓名");

name = myscanner.next();

System.out.println("请输入密码");

passwd = myscanner.next();

//比较输入的名字和密码是否正确

if("丁真".equals(name) && "666".equals(passwd)) {

System.out.println("登录成功");

break;

}chance--;

System.out.println("你还有" + chance + "次登录机会" );

}

}

}

0143 跳转控制语句-continue

基本介绍：

1. continue语句用于结束本次循环，直接进行下次循环
2. continue语句出现在多层嵌套的循环语句体中时，可以通过标签指明要跳过的是那一层循环，这个和前面的标签的使用的规则一样

基本语法：

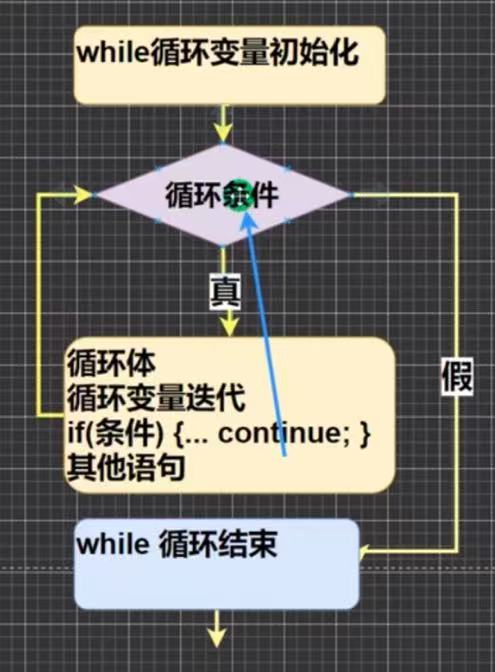
{ ……

continue;

……

}

执行流程图：



0146 跳转控制语句-return

public class return01{

public static void main(String[] args) {

for (int i = 1;i <=5;i++) {

if (i == 3) {

System.out.println("韩顺平教育" +i);

return;//当return用在方法时，表示跳出所在方法，如果使用在main，表示退出程序。此return表示跳出main

}

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

}

System.out.println("go on..");

}

}

0155 数组

数组可以存放多个同一类型的数据，数组也是一种数据类型，是引言类型。

即：数（数组）组（一组）就是一组数据

数组的使用：

1. 使用方式1-动态初始化

（1） 数组的定义：

数据类型 数组名[]=new 数据类型[大小](也可以数据类型[] 数组名)

eg: int a [] = new int[5];(int[] a)

（2） 数组的引用

数组名[下标/索引/index]

1. 使用方式2-动态初始化

（1）数组的声明：

语法：数据类型 数组名[]；也可以数据类型[] 数组名

eg: int a[];(int[] a;)

（2）创建数组

语法：数组名=new 数据类型[大小];

eg: a = new int[10];

1. 使用方式3-静态初始化(适用于元素不多)
2. 初始化数组

语法：数组类型 数组名[]=(元素值，元素值…)

1. 数组注意事项
2. 数组是多个相同类型数据的组合，实现对这些数据的同一管理
3. 数组中的元素可以是任何数据类型，包括基本类型和引用类型(字符串，接口)，但是不能混用
4. 数组创建以后，如果没有赋值，有默认值int 0,short 0 ,byte 0,long 0,float 0.0,double 0.0,char \u0000,boolean false,String null
5. 使用数组的步骤：

（1）：声明数组并开辟空间

（2）：给数组各个元素赋值

（3）使用数组

5. 数组的下标从0开始

6. 数组小标必须在指定范围内使用，即数组的下标/索引最小0，最大length-1

7. 数组属引用类型，数组型数据是对象（object）

public class array04 {

public static void main(String[] args) {

int [] nums = {4,-1,9,10,23};

int max = nums[0];

int maxindex = 0;

for ( int i =1;i < nums.length;i++) {//循环26次

if(max < nums[i]) {

max = nums[i];

maxindex = i;

}

}

System.out.println(maxindex);

}

}

0163 数组赋值机制

1. 基本数据类型赋值，这个值就是具体的数据，而不相互影响
2. 数组在默认情况下是引用传递，赋的值是地址

public class arrayassign {

public static void main(String[] args) {

int n1 = 10;

int n2 = n1;

n2 = 80;

System.out.println("n1=" + n1);

System.out.println("n2=" + n2);

int[] arr1 = {1,2,3};

int[] arr2 = arr1;

arr2[0] = 10;

for(int i = 0;i < arr1.length;i++) {

System.out.println(arr1[i]);//10 2 3

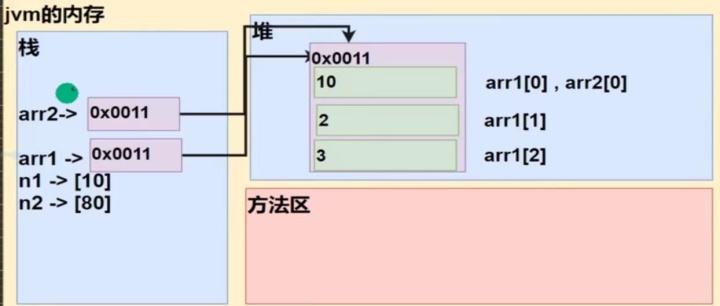
}

}

}

0164 值传递和引用传递的区别

值传递拷贝的是具体数据，引用传递拷贝的是地址。



0165 数组拷贝

public class arraycopy {

public static void main(String[] args) {

int[] arr1 = {10,20,30};

int[] arr2 = new int[arr1.length];//只要开辟空间，就会分配新地址

//遍历arr1，把每个元素拷贝到对用的位置

for(int i = 0;i < arr1.length;i++) {

arr2[i] = arr1[i];

}

arr2[0] = 100;//不会对arr1有影响

for(int i = 0;i < arr1.length;i++) {

System.out.println(arr1[i]);

}

for(int i = 0;i < arr1.length;i++) {

System.out.println(arr2[i]);

}

}

}

0166 数组反转

public class arrayreverse {

public static void main(String[] args) {

int [] arr = {11,22,33,55,66,88};

for(int i= 0;i < arr.length/2;i++) {

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

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

arr[i] = temp;

}

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

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

}

//逆序赋值法

int [] arr1 = {11,22,33,55,66,88};

int[] arr2 = new int[arr1.length];

for(int i = arr1.length-1,j=0;i >= 0;i--,j++) {

arr2[j] = arr1[i];

}

arr1 = arr2;//让arr指向arr2数据空间，此时，arr原来的数据空间就没有变量引用，会被当做垃圾销毁

for(int i = 0;i < arr1.length;i++) {

System.out.print(arr1[i] + "\t");

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

}

for(int i = 0;i < arr2.length;i++) {

System.out.print(arr2[i] + "\t");

}

}

}

0168 数组添加

import java.util.Scanner;

public class arrayadd02 {

public static void main(String[] args) {

Scanner myscanner = new Scanner(System.in);

//因为用户什么时候退出不知道，使用do-while + break来控制

int[] arr = {1,2,3};

do{

int[] arrnew = new int[arr.length+1];

for ( int i =0;i < arr.length;i++) {//循环26次

arrnew[i] = arr[i];

}

System.out.println("请输入你要添加的元素");

int addnum = myscanner.nextInt();

arrnew[arr.length-1] = addnum;

arr = arrnew;//让arr指向arrnew，arr = arrnew，将arrnew的地址赋给arr，没有对象指向原来arr的地址，那么原来arr数组就被销毁

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

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

}

System.out.println("是否继续添加 y/n");

char key = myscanner.next().charAt(0);

if(key == 'n') {

break;

}

}while(true);

}

}

0170 排序

排序是将多个数据，依指定的顺序进行排列的过程

排序的分类：

1. 内部排序：

指将需要处理的所有数据都加载到内部存储器中进行排序。包括（交换式排序法，选择式排序法和插入式排序法）；

1. 外部排序法

数据量过大，无法全部加载到内存中，需要借助外部存储进行排序。包括（合并排序法和直接合并排序法）

冒泡排序法基本思想：通过对待排序序列从后往前，一次比较相邻元素的值，若发现逆序则交换，使值较大的元素逐渐从前移向后部。

冒泡排序特点：

1. 共有n个元素，进行n-1轮排序
2. 每一轮排序可以确定一个数的位置，比如第一轮排序确定最大数
3. 当进行比较时，如果前面的数大于后面的数，就交换
4. 每轮参与比较的数较少，第m轮比较n-m个数

public class bubblesort {

public static void main(String[] args) {

int [] nums = {24,69,80,57,13};

int max = nums[0];

for (int i = 0;i < nums.length-1;i++) {

for(int j =0;j < nums.length-i-1;j++) {

if (nums[j] > nums[j+1]) {

max = nums[j];

nums[j] = nums[j+1];

nums[j+1] = max;

}

}

System.out.println("第" + (i+1) + " 轮排序结果为：");

for (int m = 0;m < nums.length;m++){

System.out.print(nums[m] + "\t");

}

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

}

for ( int i =0;i < nums.length;i++) {//循环26次

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

}

}

}

0174 查找

常用的查找有两种：

1. 顺序查找
2. 二分查找

import java.util.Scanner;

public class seqsearch {

public static void main(String[] args) {

//定义一个字符串数组

String[] names = {"一","二","三","五"};

Scanner myscanner = new Scanner(System.in);

System.out.println("请输入名字");

String findname = myscanner.next();

//遍历数组，主义比较，如果有，则提示信息，并退出

int index = -1;

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

if(findname.equals(names[i])) {

System.out.println("找到" + findname);

System.out.println("下标为=" + i);

index = i;

break;

}

}

if(index == -1) {

System.out.println("没找到" + findname);

}

}

}

0175 多维数组-二维数组

关于二维数组的关键概念

1. 二维数组元素个数可用数组名.length统计
2. 二维数组的每一个元素是一维数组，所以如果需要得到每一个一维数组的值，还需要再次遍历
3. 如果要访问第（i+1）个一维数组的第j+1个值，arr[i][j]

0176 二维数组的使用

1. 使用方式1-动态初始化
2. 语法：类型[][] 数组名=new 类型[大小][大小]

public class twodimensionalarray02 {

public static void main(String[] args) {

int arr[][] = new int[2][3];

arr[1][2] = 8;

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

//遍历二维数组的每个元素（数组）

//arr[i].length得到对应的每一个一维数组的长度

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

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

}

System.out.println();

}

}

}

1. 二维数组内存形式：
2. 使用方式2-动态初始化
3. 先声明：类型 数组名[][]
4. 在定义（开辟空间）数组名=new 类型[大小][大小]
5. 赋值（有默认值，比如int 类型的就是0）
6. 使用方式3：动态初始化-列数不确定

每一个元素的个数不同

public class twodimensionalarray03 {

public static void main(String[] args) {

int arr[][] = new int[3][];

for(int i = 0;i < arr.length;i++) {//遍历二维数组的每个元素（数组）

//给每个一维数组开空间，如果没有给一维数组new，那么arr[i]就是null

arr[i] = new int[i+1];

//遍历一维数组，并给一维数组的每个元素赋值

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

arr[i][j] = i + 1;

}

}

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

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

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

}

System.out.println("\n");

}

}

}

1. 使用方式4：动态初始化-列数不确定
2. 定义 类型 数组名[][] = {{值1，值2…}，{值1，值2…},{值1，值2…}}
3. 使用即可[固定方式访问]

//杨辉三角

public class yanghui {

public static void main(String[] args) {

int yanghui[][] = new int[10][];

for(int i = 0;i < yanghui.length;i++) {//遍历二维数组的每个元素（数组）

//给每个一维数组开空间，如果没有给一维数组new，那么arr[i]就是null

yanghui[i] = new int[i+1];

//遍历一维数组，并给一维数组的每个元素赋值

for(int j = 0;j < yanghui[i].length;j++) {

if (j == 0 ||j == yanghui[i].length - 1){

yanghui[i][j] = 1;

} else {

yanghui[i][j] = yanghui[i-1][j] + yanghui[i-1][j-1];

}

}

}

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

for(int j = 0;j < yanghui[i].length;j++) {

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

}

System.out.println();

}

}

}

0177 二维数组使用细节和注意事项

1. 一维数组的声明方式有：

int[] x或者int x[]

1. 二维数组的声明方式有:

int[][] y 或者int[] y[] 或者 int y[][]

1. 二维数组本质上是由多个一维数组组成的，它的各个一维数组的长度可以相同，也可以不同

Int[] x,y[];以下选项通过编译器的是( )：

a)x[0] = y;//错误，不能将一个二维数组赋给一个int

b)y[0] = x;//可以,讲一个一维数组赋给另一个一维数组

c)y[0][0] = x;//错误，把一个一维数组赋给一个int

d)x[0][0] = y;//错误，x是一维数组

e)y[0][0] = x[0];//可以，把一个int复制给一个int

f)x= y;//把一个二维数组赋给一个一维数组

[注]:x是一个整形一维数组，y是一个整形二维数组

0191 java 类与对象

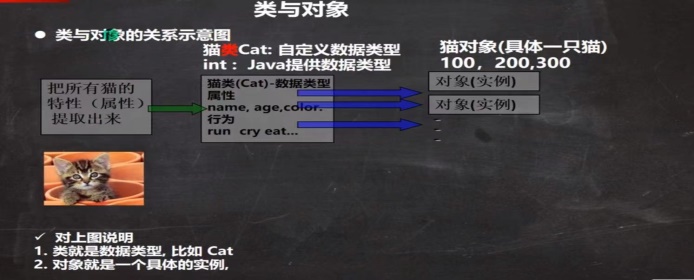
1)单独的定义变脸解决

2)使用数组解决

使用现有技术解决不利于数据的管理，引入类与对象(oop)

对象[属性，行为]

类与对象的关系示意图：



快速入门：

public class object01 {

public static void main(String[] args) {

//1.new Cat()创建一只猫

//2.Cat cat1 = new Cat() 把创建的猫赋给cat1

Cat cat1 = new Cat();

cat1.name = "小白";

cat1.age = 3;

cat1.color = "白色";

Cat cat2 = new Cat();

cat2.name = "小花";

cat2.age = 100;

cat2.color = "花色";

//访问对象属性

System.out.println("第1只猫信息" + cat1.name + " " + cat1.age + " " + cat1.color);

System.out.println("第2只猫信息" + cat2.name + " " + cat2.age + " " + cat2.color);

}

}

class Cat {

String name;

int age;

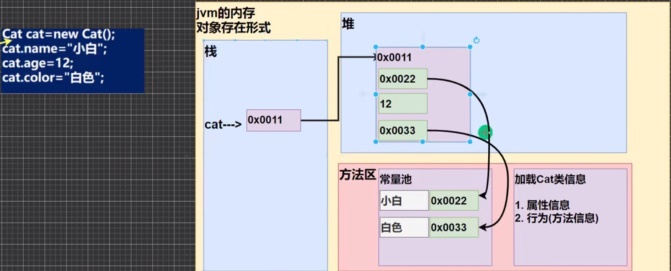
String color;

}

类与对象的区别：

1. 类是抽象的，概念的，代表一类事物
2. 对象是具体的，实际的，代表一个具体事物
3. 类是对象的模板，对象是类的一个个体，对应一个实例

0194 对象内存布局



整形是基本数据类型，直接存在堆中，不用放在常量池

Cat是对象引用，堆中存放的是真正的对象。

0195 属性概念

1. 从概念或叫法上看，成员变量=属性=field(字段)
2. 属性是类的组成部分，一般是基本数据类型，也可以是引用类型(对象，数组)

注意事项和使用细节：

1. 属性的定义语法同变量，示例：访问修饰符 属性类型 属性名；

有四种访问修饰符(控制属性的访问范围):public,proctected,默认,private

1. 属性的定义类型可以为任意类型，包括基本类型或引用类型
2. 属性如果不赋值，有默认值，规则和数组一致

int 0,short 0 ,byte 0,long 0,float 0.0,double 0.0,char \u0000,boolean false,String null

public class object02 {

public static void main(String[] args) {

Person p1 = new Person();

//访问对象属性

System.out.println("\n当前这个人的信息");

System.out.println("age=" + p1.age + "name=" + p1.name + "sal=" + p1.sal + "ispass=" + p1.ispass);

}

}

class Person {

int age;

String name;

double sal;

boolean ispass;

}

0197 创建对象

1. 先声明再创建

Cat cat;//声明对象cat，cat是null

cat = new Cat();//创建空间，分配地址，并把地址填回栈

1. 直接声明

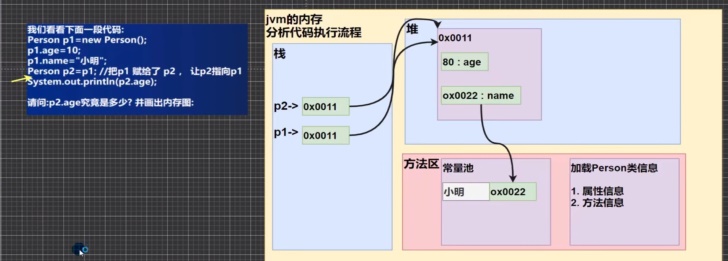
Cat cat = new Cat()

访问对象：

基本语法：

对象名.属性名；

0198 类与对象的分配机制



Java内存的结构分析

1. 栈：一般存放基本数据类型（局部变量）
2. 堆：存放对象（Cat cat,数组等）
3. 方法区：常量池（常量，比如字符串），类加载信息

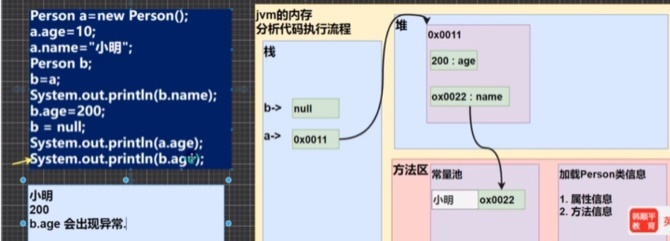
Java创建对象的简单流程分析

Person p1 = new Person();

p1.name = “jack”;

p1.age = 10;

1. 先加载Person类属性（属性和方法信息，只会加载一次）
2. 在堆中分配空间（空间大小由属性决定），进行默认初始化（看规则）
3. 把地址赋给p1, p1就指向对象（堆中）
4. 指定初始化：p1.name = “jack”;p1.age = 10;



b = null;相当于将b指向堆中对象的地址线切断

0201 成员方法

在某些情况下，我们还需要定义成员方法（简称方法）。比如人类的一些行为：学习。

public class method {

public static void main(String[] args) {

//方法使用

//1.方法写好后，不调用，不执行

//2.先创建对象，然后调用方法即可

//3.

Person p1 = new Person();

p1.speak();//调用方法

p1.cal01();

p1.cal02(5);//调用cal02方法,同时n=5

//调用getsum方法，同时num1 = 10，num2=20

//把方法getsum返回的值赋给变量returnres

int returnres = p1.getsum(10,20);

System.out.println(returnres);

}

}

class Person {

String name;

int age;

//方法(成员方法)

//添加speak成员方法，输出我是一个好人

//1.public:表示方法公开

//2.void:表示方法没有返回值

//3.speak()：speak是方法名，()是形参列表

//4.{}方法体，可以写我们要执行的代码

//5.System.out.println("我是一个好人");表示我们的方法就是要输出一句话

public void speak() {

System.out.println("我是一个好人");

}

//添加cal01成员方法，可以计算从1+..+100的结果

public void cal01() {

//循环完成

int res = 0;

for(int i = 1;i <= 1000;i++) {

res += i;

}

System.out.println("cal01计算结果=" + res);

}

//添加cal02成员方法，该方法可以接收一个数n,计算从1+..+n的结果

//1.

//2.

//3.

public void cal02(int n) {

//循环完成

int res = 0;

for(int i = 1;i <= n;i++) {

res += i;

}

System.out.println("cal02计算结果=" + res);

}

//添加getsum成员方法，可以计算两个数的和

//1.public:表示方法公开

//2.int:表示方法执行后，返回一个int值

//3.getsum()方法名

//4.(int num1,int num2)形参列表，2个形参，可以接收用户传传入的两个值

//5.return res;表示把res的值返回

public int getsum(int num1,int num2) {

//循环完成

int res = num1 + num2;

return res;

}

}

方法的调用机制：

当 return语句执行后，getsum这个栈就被清理了

方法调用小结：

1. 当程序执行到方法时，就会开辟一个独立的空间（栈空间）
2. 当方法执行完毕，或者执行到return语句时，就会返回
3. 返回到调用方法的地方
4. 返回后，继续执行方法后面的代码。
5. main方法（栈）执行完毕，相当于整个程序退出

成员方法的必要性：

传统的遍历数组方法，多次遍历时，代码冗余度太高

public class method02 {

public static void main(String[] args) {

int [][] map = {{0,0,1},{1,1,1},{1,1,3}};

Mytools tool = new Mytools();

tool.printarr(map);

}

}

class Mytools {

public void printarr(int[][] map1) {

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

//对传入的map1数组进行遍历输出

for (int i = 0;i < map1.length;i++) {

for (int j = 0;j < map1[i].length;j++) {

System.out.print(map1[i][j] + " ");

}

System.out.println();

}

}

}

0205 成员方法的定义

访问修饰符（如public）返回数据类型 方法名 （形参列表 ）{

//方法体语句；

return 返回值；

}

1. 形参列表：表示成员方法输入：cal02(int n)，getsum(int num1,int num2)
2. 返回数据类型：表示成员方法输出，void表示没有返回值
3. 方法主体：表示为了实现某一功能代码块
4. return语句不是必须的

0206 方法的注意事项和细节

1. 访问修饰符（作用是控制方法的使用范围）

如果不写表示默认访问，有四种[public,protected,默认,private]

1. 返回数据类型
2. 一个方法最多有一个返回值，如果要返回多个结果，可返回数组
3. 返回类型可以为任意类型，包含基本类型或引用类型（数组，对象）
4. 如果方法要求返回数据类型，则方法体中最后的执行语句必须为return值；而且要求返回值类型必须和return的值类型一致或兼容
5. 如果方法时void，则方法体中可以没有return语句，或者只写return;
6. 在实际工作中，遵循驼峰命名法，方法名要有一定含义，最好见名知意
7. 形参列表
8. 一个方法可以有0个参数，也可以有多个参数，中间用逗号隔开，比如getsumandsub(int n1,int n2)
9. 调用带参数的方法时，一定对应着参数列表传入相同类型或兼容类型的参数
10. 方法定义时的参数称为形式参数，调用方法调用时实际传入的参数称为实际参数，形参和实参的类型要一致或兼容，个数，顺序必须一致

public class methoddetail {

public static void main(String[] args) {

AA a = new AA();

int[] arr1 = a.getsumandsub(3,6);

System.out.println("和=" + arr1[0]);

System.out.println("差=" + arr1[1]);

a.f3("abc",10);

}

}

class AA {

public int[] getsumandsub(int n1,int n2) {

int[] arr = new int[2];//创建一个数组

arr[0] = n1 + n2;

arr[1] = n1 - n2;

return arr;

}

public double f1() {

int d1 = 1;

return d1; //int ->double可以自动转换；double->int不可以

}

public void f2() {

System.out.println("hello");

int n = 10;

return;

}

public void f3(String str,int n) {

}

}

1. 方法体
2. 方法不能嵌套定义
3. 里面写完功能的具体的语句，可以为输入，输出，变量，运算，分支，循环，方法调用

0208 方法调用细节说明

1. 同一个类中的方法调用：直接调用即可
2. 跨类中的方法A类调用B类方法：需要通过对象名调用
3. 特别说明，跨类的方法调用和方法的访问修饰符相关

//跨类中的方法A类调用B类方法

public class methoddetail02 {

public static void main (String[] args) {

A a = new A();

a.sayok();

a.m1();

}

}

class A {

public void print(int n) {

System.out.println("print()方法被调用 n=" + n);

}

public void sayok() { //sayok调用print(直接调用即可)

print(10);

System.out.println("继续执行sayok()~~~");

}

}

//跨类中的方法A调用B类方法：需要通过对象名调用

//执行一个方法就开辟一个栈

public void m1() {

//创建B对象，然后再调用方法即可

System.out.println("m1()方法被调用");

B b = new B();

b.hi;

System.out.println("m1()继续执行");

}

class B {

public void hi() {

System.out.println("B类中的hi()被执行");

}

}

//判断奇偶性

public class methodexercise {

public static void main(String[] args) {

AA aa = new AA();

if(aa.is0dd(20)) {

System.out.println("实奇数");

} else {

System.out.println("实偶数");

}

}

}

class AA {

public boolean is0dd(int num) {

// if(num % 2 !=0 ) {

// return true;

// } else {

// return false;

// }

// return num % 2 !=0 ? true; false;

return num % 2 !=0;

}

}

//打印字符

public class methodexercise02 {

public static void main(String[] args) {

AA aa = new AA();

aa.printchar(9,10,'\*');

}

}

class AA {

public void printchar(int row,int col,char c1) {

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

for (int j =0;j < col;j++) {

System.out.print(c1);

}

System.out.println();

}

}

}

0210 方法传参机制

1. 基本数据类型的传参机制

public class methodparameter01 {

public static void main(String[] args) {

int a = 10;

int b = 20;

AA aa = new AA();

aa.swap(a,b);//一旦调用方法就会产生新栈

System.out.println("主方法a=" + a + " b=" + b);//10 20

}

}

class AA {

public void swap(int a,int b) {

System.out.println("\na和b交换前的值\na=" + a + " b=" + b);//10 20（输出的是swap栈里的a,b）

int tmp = a;

a = b;

b = tmp;

System.out.println("\na和b交换前的值\na=" + a + " b=" + b);//20 10（输出的是swap栈里的a,b）

}

}

基本数据类型，传递的是值（值拷贝），形参的任何改变不影响实参

1. 引用数据类型的传参机制

引用类型传递的是地址（传递的也是值，但值是地址），可以通过形参影响实参

public class methodparameter02 {

public static void main(String[] args) {

AA a = new AA();

int[] arr = {1,2,3};

a.testarr(arr);

System.out.println("main的arr数组");

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

System.out.println(arr[i] + "\t");

}

System.out.println();

}

}

class AA {

public void testarr(int[] arr) {

arr[0] = 200;

System.out.println("testarr的arr数组");

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

System.out.println(arr[i] + "\t");

}

System.out.println();

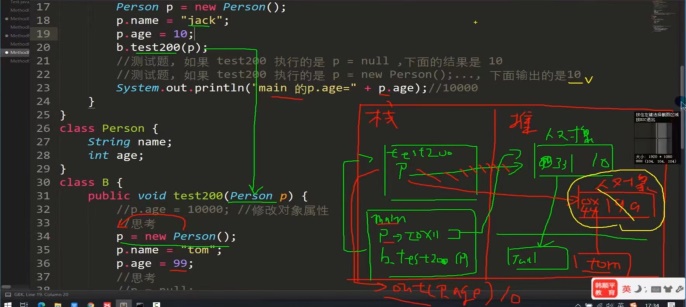
}

}

p = null;会将形参指向对象的线置空

p= new Person()会在堆中创建一个新对象。

内存机制分析图：



//clone对象

public class methodexercise03 {

public static void main(String[] args) {

Person p1 = new Person();

p1.name = "mainn";

p1.age = 100;

Mytools tools = new Mytools();

Person p2 = tools.copyperson(p1);

//到此p1,p2是两个独立的对象，属性相同

System.out.println("p1的属性 age=" + p1.age + "名字=" + p1.name);

System.out.println("p2的属性 age=" + p2.age + "名字=" + p2.name);

//可以通过对象比较看看是否为一个对象

System.out.println(p1 == p2);

}

}

class Person {

String name;

int age;

}

class Mytools {

//编写思路

//1.方法的返回类型

//2.方法的名字

//3.方法的形参

//4.方法体，创建一个新的对象，并复制属性返回即可

public Person copyperson(Person p) {

Person p2 = new Person();

//把原来对象的属性值赋给新的对象

p2.name = p.name;

p2.age = p.age;

return p2;

}

}

内存机制分析图：

0214 递归调用

递归调用的本质是方法调用

public class recursion {

public static void main(String[] args) {

T t = new 2oT();

t.test(5);

}

}

class T {

public void test(int n) {

if (n > 2 ) {

test(n-1);

}

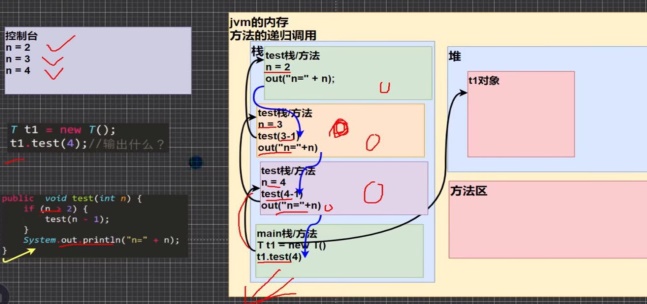
System.out.println("n=" + n);

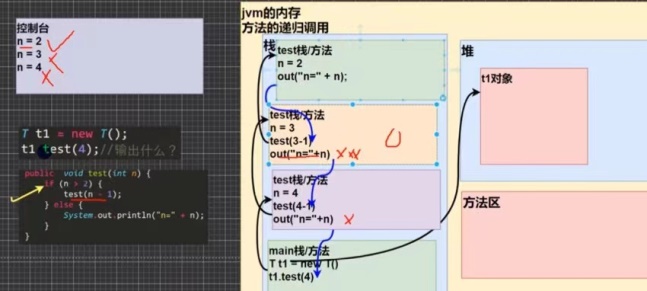
}

}

内存机制分析图：

每一个栈都完整的执行一个方法





递归重要规则：

1. 执行一个方法时，就创建一个新的受保护的独立空间（栈空间）
2. 方法的局部变量是独立的，不会相互影响，比如n变量
3. 如果方法中使用的是引用类型变量（比如数组，对象），就会共享该引用类型的数据
4. 递归必须向退出递归的条件逼近否则就是无限递归
5. 当一个方法执行完毕或者遇到return，就会返回，遵守谁调用，就将结果返回给谁，同时当方法执行完毕或者返回时，该方法也就执行完毕

//桃子总数

public class recursionexercise02 {

public static void main(String[] args) {

T t = new T();

int n = 8;

int peachnum = t.peach(n);

if (peachnum != -1) {

System.out.println("第" + n + "天有" + peachnum + "个桃子");

}

}

}

class T {

public int peach(int n) {

if (n ==10) {

return 1;

} else if (n >=1 && n <=9 ) {

return((peach(n+1)+1)) \* 2;

} else {

System.out.println("要求输入的n在1-10");

return -1;

}

}

}

//迷宫

public class migong {

public static void main(String[] args) {

int[][] map = new int[8][7];

for (int i = 0;i < 7;i++) {

map[0][i] = 1;

map[7][i] = 1;

}

for (int i = 0;i < 7;i++) {

map[i][0] = 1;

map[i][6] = 1;

}

map[3][1] = 1;

map[3][2] = 1;

//使用findway

T t = new T();

t.findway(map,1,1);

System.out.println("当前地图情况===========================================");

for (int i = 0;i < 7;i++) {

for (int j = 0;j < 7;j++) {

System.out.print(map[i][j] + "\t");

}

System.out.println();

}

}

}

class T {

public boolean findway(int[][] map,int i,int j) {

//1.findway方法专门用来找出迷宫的路径

//2.如果找到，就返回true，否则返回false

//3.map就是二维数组，即表示迷宫

//4.i,j就是老鼠的位置，初始化的位置为（1,1）

//5.规定map数组的各个值的含义：0表示可以走，1表示障碍物，2表示可以走t通，3表示走过，但是走不通是死路

//6.当map[6][5] == 2就说明找到通路，就可以结束，否则就继续找

//7.确定老鼠找路策略：下->右->上->左

if (map[6][5] == 2) {

return true;

} else {

if(map[i][j] == 0) {//当前这个位置为0，说明表示可以走

//假定可以走通

map[i][j] = 2;

//使用找路策略，来确定该位置是否真的可以走通

//下->右->上->左

if(findway(map,i+1,j)) {

return true;

} else if(findway(map,i,j+1)) {

return true;

} else if(findway(map,i-1,j)) {

return true;

} else if(findway(map,i,j-1)) {

return true;

} else {

map[i][j] = 3;

return false;

}

} else {

//map[i][j] == 1,2,3

return false;

}

}

}

}

//汉诺塔

public class hanluotower {

public static void main(String[] args) {

Tower t = new Tower();

t.move(3,'a','b','c');

}

}

class Tower {

public void move (int num,char a,char b,char c) {

if(num == 1) {

System.out.println(a + "->" + c);

} else {

//如果有多个盘，可以看成两个，最下面和上面的所有盘（num-1）

//(1)先移动上面所有的盘到b，借助c

move(num-1,a,c,b);

//(2)把最下面的这个盘，移动到c

System.out.println(a + "->" + c);

//(3)把b塔的所有盘，移动到c，借助a

move(num-1,b,a,c);

}

}

}

0227 方法重载（overload）

Java 中允许同一个类中，多个同名方法的存在，但要求形参列表不一致。比如System.out.println(),out是PrintStream类型

1）减轻了起名的麻烦

2）减轻了记名的麻烦

方法重载注意事项和使用细节：

1. 方法名：必须相同
2. 形参列表：必须不同（形参类型或个数或顺序，至少有一样不同，参数名无要求）
3. 返回类型：无要求

0233 可变参数

Java允许将同一个类中多个同名同功能但参数个数不同的方法，封装成一个方法

基本语法：

访问修饰符 返回类型 方法名（数据类型…形参名） {

}

1.int...表示接受的是可变参数，类型是int，即可以接受多个int(0-多)

2.使用可变参数可以当做数组来使用，即nums可以当做数组

public class varparameter01 {

public static void main(String[] args) {

Methods t = new Methods();

System.out.println(t.sum(1,2,100));

}

}

class Methods {

//使用可变参数

//1.int...表示接受的是可变参数，类型是int，即可以接受多个int(0-多)

//2.使用可变参数可以当做数组来使用，即nums可以当做数组

public int sum(int... nums) {

int res = 0;

System.out.println("接受参数个数=" + nums.length);

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

res += nums[i];

}

return res;

}

}

可变参数细节：

1. 可变参数的实参可以是0或多个
2. 可变参数的实参可以是数组
3. 可变参数的本质就是数组
4. 可变参数可以和普通类型的参数放在形参列表，但必须保证可变参数在最后
5. 一个形参列表中最多只能有一个可变参数

0236 作用域

1. 在java 编程中，主要的变量就是属性（成员变量）和局部变量
2. 局部变量一般是指在成员方法中定义的变量，也就是除了属性之外的其他变量，作用域为定义它的代码块中
3. 全局变量也就是属性，作用域为整个类体
4. 全局变量即属性在定义时，可直接赋值
5. 全局变量（属性）可以不赋值，直接使用，因为有默认值，局部变量必须赋值后，才能使用，因为没有默认值。

0237 作用域使用细节

1. 属性和局部变量可以重名，访问时遵循就近原则
2. 在同一个作用域中，比如在同一个成员方法中，两个局部变量，不能重名
3. 属性声明周期较长，伴随着对象的创建而创建，伴随着对象的销毁而销毁，局部变量，生命周期较短，伴随着它的代码块的执行而创建，伴随着代码块的结束而销毁。即在一次方法调用过程中
4. 作用域不同

全局变量/属性：可以被本类使用，或其他类使用（通过对象调用）

局部变量：只能在本类中对应的方法中调用

1. 修饰符不同

全局变量/属性可以加修饰符

局部变量不可以加修饰符

0239 构造方法/构造器

构造方法又叫构造器，是类的一种特殊方法，它的主要作用是完成对新对象的初始化。

基本语法

[修饰符] 方法名（形参列表）{

方法体；

}

说明：

1. 构造器的修饰符可以默认，也可以是public protected private
2. 构造器没有返回值
3. 方法名和类名字必须一样
4. 参数列表和成员方法一样的规则
5. 构造器的调用由系统完成

构造方法/构造器注意事项和使用细节

1. 一个类可以定义多个不同的构造器，及构造器重载
2. 构造器名要和类名相同
3. 构造器没有返回值
4. 构造器是完成对象的初始化，并不是创建对象
5. 在创建对象时，系统自动的调用该类的方法
6. 如果程序员没有定义构造器，系统会自动给类生成一个默认无参参构造器

通过反编译查看（将class文件转成源文件）：使用javap 类名.class(class可省略)

常用 javap –c /–v 类名

1. 一旦定义了自己的构造器，默认构造器就覆盖了，就不能再使用默认的无参构造器，除非显示的定义一下，即：Dog（）{}写

public class constructor01 {

public static void main(String[] args) {

//当我们new一个对象时，直接通过构造器指定名字和年龄

Person t1 = new Person("smith",80);

Person t2 = new Person("tom");

System.out.println("t对象的name" + t1.name);

System.out.println("t对象的age" + t1.age);

System.out.println("t对象的name" + t2.name);

System.out.println("t对象的age" + t2.age);

//t1.person()调用构造器是不行

Dog d1 = new Dog();//显示定义无参构造器了可以使用

}

}

class Dog {

//如果程序员没有定义构造器，系统会自动给类生成一个默认无参参构造器

/\* Dog() {

}\*/

public Dog(String dname) {

//...

}

Dog() {//显示定义一下无参构造器

}

}

class Person {

String name;

int age;

//1.构造器没有返回值，也不能写void

//2.构造器的名称和类person一样

//3.(String pname,int page)是构造器形参列表，规则和成员方法一样

public Person(String pname,int page) {

System.out.println("构造器被调用,完成对象属性的初始化");

name = pname;

age = page;

}

public Person(String p1name) {

System.out.println("构造器2完成对象属性的初始化");

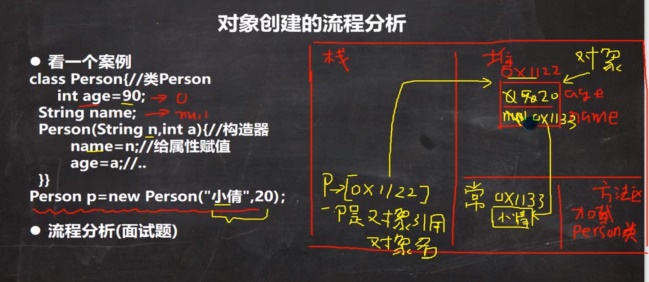
name = p1name;

}

}

0244 对象创建的流程分析

1. 加载Person类信息（Person.class），只会加载一次
2. 在堆中分配空间（地址）
3. 完成对象初始化
   1. 默认初始化age = 0,name = null
   2. 显示初始化age = 90,name = null
   3. 构造器初始化age =20,name = 小倩
   4. 把对象在堆中的地址，返回给p



0245 this

Java虚拟机会给每个对象分配this，代表当前对象

public class this01 {

public static void main(String[] args) {

Person t = new Person("大壮",17);

System.out.println("t对象的name=" + t.name);

System.out.println("t对象的age=" + t.age);

}

}

class Person {

String name;

int age;

public Person(String name,int age) {

System.out.println("构造器被调用,完成对象属性的初始化");

this.name = name;

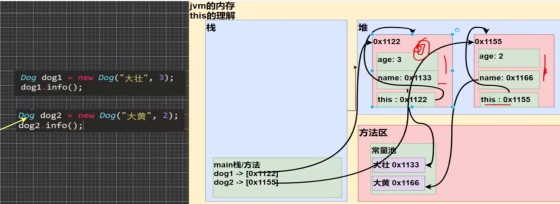
this.age = age;

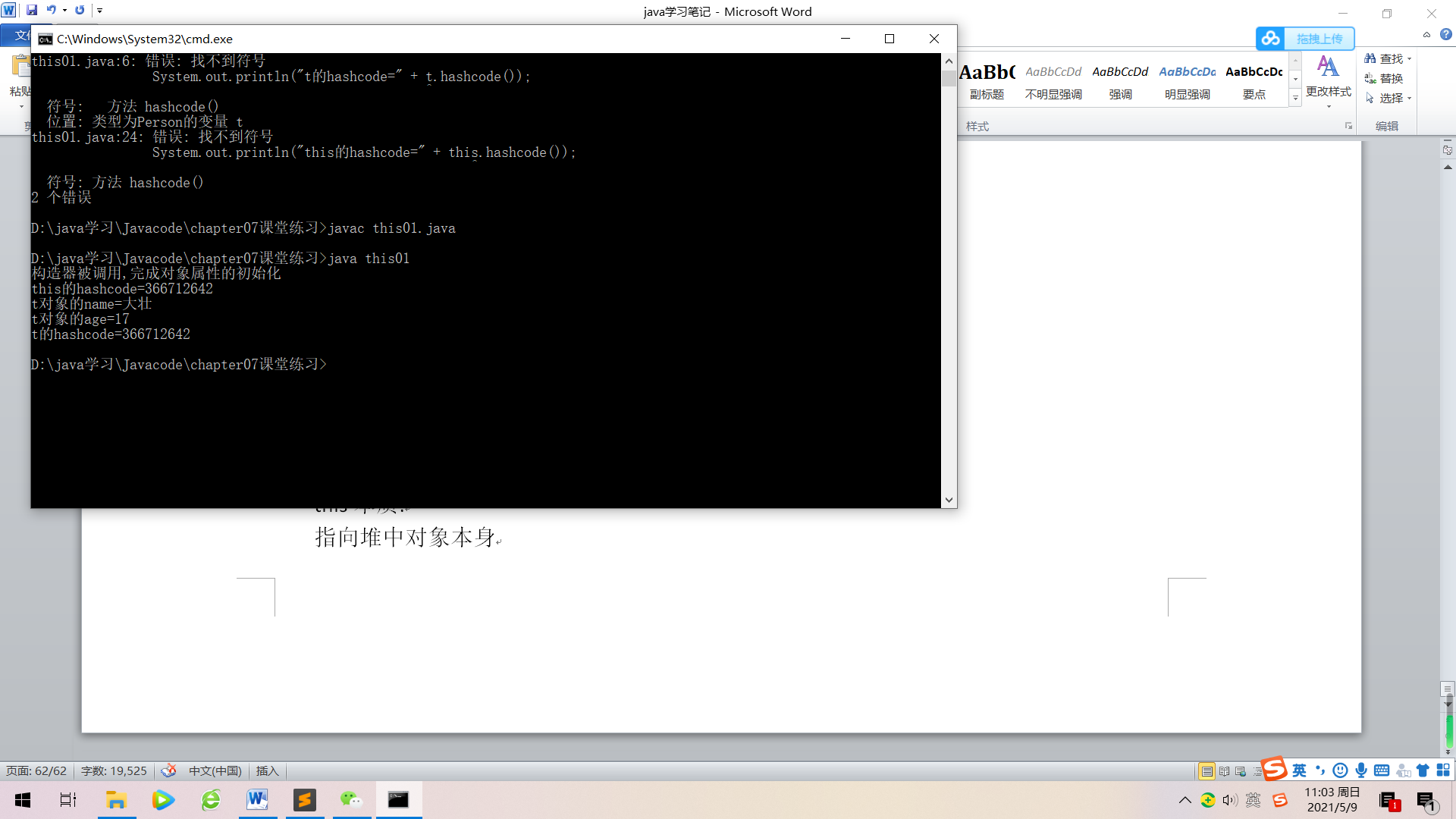
}

}

this本质:

指向堆中对象本身，简单说，那个对象调用，就指向那个对象





this的注意事项和使用细节

1. this关键字可以用来访问本类的属性，，方法，构造器
2. this用于区分当前类的属性和局部变量
3. 访问成员方法的语法：this.方法名（参数列表）；
4. 访问构造器语法：this（参数列表）；注意只能在构造器中使用且this（参数列表），必须放在第一句
5. This不能在定义的外部使用，只能在类定义的方法中使用

public class thisdetail {

public static void main(String[] args) {

// T t1 = new T();

// t1.f2();

T t2 = new T();

t2.f3();

}

}

class T {

String name = "jack";

int num = 100;

public T() {

this("jack",100);//看到这句话去访问T(String name,int age)构造器

System.out.println("T()构造器");

}

public T(String name,int age) {

System.out.println("T(String name,int age)构造器");

}

public void f1() {

System.out.println("f1()方法..");

}

public void f2() {

System.out.println("f2()方法..");

//调用本类f1

//第一种方式

f1();

//第二种方式

this.f1();

}

//this关键字可以用来访问本类属性

public void f3() {

String name = "smith";

//传统方式

System.out.println("name=" + name + "num" + num);//如果有局部变量就访问局部变量，没有就找属性

//使用this访问属性

System.out.println("name=" + this.name + "num" + this.num);//不管有没有有局部变量访问的都是属性

}

}

//判断两个人的属性是否相同

public class thisexercise {

public static void main(String[] args) {

Person t1 = new Person("marry",20);

Person t2 = new Person("smith",30);

System.out.println(t1.compareto(t2));

}

}

class Person {

String name;

int age;

//构造器

public Person(String name,int age) {

this.name = name;

this.age = age;

}

public boolean compareto(Person p) {

return this.name.equals(p.name) && this.age == p.age;

}

}

public T(String name,int age) {

System.out.println("T(String name,int age)构造器");

}

public void f1() {

System.out.println("f1()方法..");

}

public void f2() {

System.out.println("f2()方法..");

//调用本类f1

//第一种方式

f1();

//第二种方式

this.f1();

}

//this关键字可以用来访问本类属性

public void f3() {

String name = "smith";

//传统方式

System.out.println("name=" + name + "num" + num);//如果有局部变量就访问局部变量，没有就找属性

//使用this访问属性

System.out.println("name=" + this.name + "num" + this.num);//不管有没有有局部变量访问的都是属性

}

}

//判断两个人的属性是否相同

public class thisexercise {

public static void main(String[] args) {

Person t1 = new Person("marry",20);

Person t2 = new Person("smith",30);

System.out.println(t1.compareto(t2));

}

}

class Person {

String name;

int age;

//构造器

public Person(String name,int age) {

this.name = name;

this.age = age;

}

public boolean compareto(Person p) {

return this.name.equals(p.name) && this.age == p.age;

}

}

面向对象编程

0263 ：

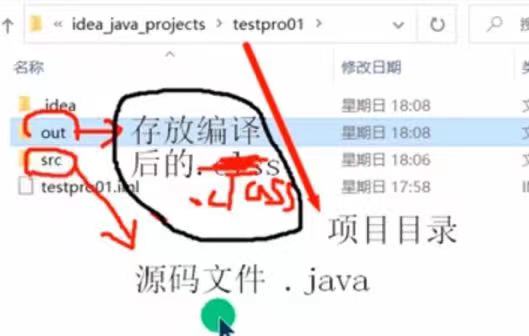
IDE（集成开发环境）-IDEA

1. IDEA全称IntelliJ IDEA

IDE（集成开发环境）-Eclipse

1. 是一个开放源代码的，基于java的可扩展开发平台

0264 IDEA使用



IDEA常用快捷键

1. 删除当前行：默认ctrl + Y
2. 复制当前行：默认ctrl + D
3. 代码补全：alt + /
4. 添加注释和取消注释ctrl + /[第一次是添加注释，第二次是取消注释]
5. 导入该行需要的类：
6. 快速格式化代码：ctrl + alt + L
7. 快速运行程序：alt + R
8. 生成构造器等：alt + insert
9. 查看一个类的层级关系 ctrl +H
10. 将光标放在一个方法上，输入Ctrl + B，可以定位到方法
11. 自动的分配变量名，通过在后面加.var

IDEA模板：

File ->settings ->editor ->Live templates->

模板可以高效的完成开发，提高速度

0272 包

包的三大作用：

1. 区分相同名字的类
2. 当类很多时，可以很好的管理类
3. 控制访问范围

包的基本语法：

package com.hspedu;

说明：

1. package关键字,表示打包
2. com.hspedu：表示包名

包的本质分析：

包的本质实际上就是创建不同的文件夹/目录来保存类文件



可将同一个类放在两个不同的包下面

包的命名:

命名规则：

只能包含数字、字母、下划线、小圆点，但不能用数字开头，不能是关键字或保留字划线

命名规则：

一般是小写字母+小圆点，一般是：

com.公司名.项目名.业务模块名

常用的包：

一个包下，包含很多的类，java中常用的包有：

1. java.lang: //lang是基本包，默认引入，不需要再引入
2. java.util: //util包，系统提供的工具包，工具类，使用Scanner
3. java.awt: //是做java的界面开发，GUI

包的使用细节：

语法：import 包

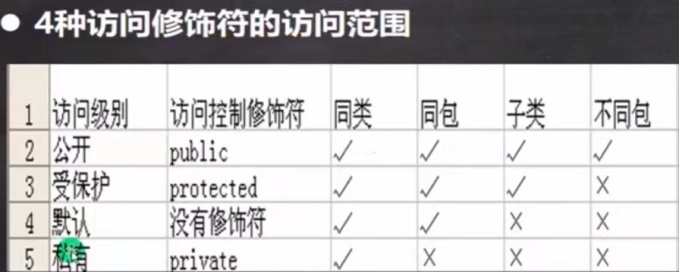
1. package的作用是声明当前类所在的包，需要放在类的最上面，一个类中最多一句package
2. import 指令放在package的下面，在类定义前面，可以有多句且没有顺序要求

0278： 访问修饰符

Java提供四种访问控制修饰符号，用于控制方法和属性（成员变量）的访问权限（范围）；

1. 公开级别：用public修饰，对外公开
2. 受保护级别：用protected修饰，对子类和同一个包中的类公开
3. 默认级别：没有访问修饰号，向同一个包的类公开
4. 私有级别：用private修饰，只有类本身可以访问，不对外公开

4种访问修饰符的访问范围：



访问修饰符的注意事项：

1. 修饰符可以用来修饰类中的属性，成员方法以及类
2. 只有默认的和public才能修饰类，并且遵循上述访问权限的特点
3. 成员方法的访问规则和属性完全一样

面向对象编程：

1. 封装：

把抽象出的数据[属性]和对数据的操作[方法]封装在一起，数据被保护在内部，程序的其他部分只有通过被授权的操作[方法],才能对数据进行操作

封装的理解和好处：

1. 隐藏实现细节：方法（连接数据库） <--调用（传入参数…）
2. 可以对数据进行验证，保证安全合理

封装实现的步骤：

1. 将属性进行私有化private（不能直接修改属性）
2. 提供一个公共的（public）set方法，用于对属性判断并赋值

Public void setXxx（类型 参数名）{Xxx表示某个属性

//加入数据验证的业务逻辑

属性 = 参数名；

}

1. 提供一个公共的（public）get方法，用于获取属性的值

Public 数据类型 getXxx（） {//权限判断，Xxx某个属性

return xx;

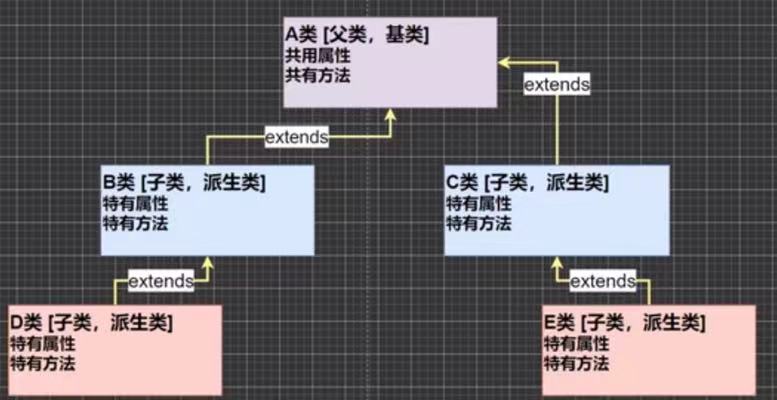
}

0285 继承extends

两个类的属性和方法很多都是一样的，可以使用方法继承，避免代码的重复性。

继承可解决代码重复，当多个类存在相同的属性（变量）和方法时，可以从这些类中抽象出父类，在父类中定义这些相同的属性和方法，所有的子类不需要重新定义这些属性和方法，只需要通过extends来声明继承父类即可

继承原理图：



基本语法：

class 子类 extends 父类 {

}

1. 子类就会自动拥有父类定义的属性和方法
2. 父类又叫超类，基类
3. 子类又叫派生类

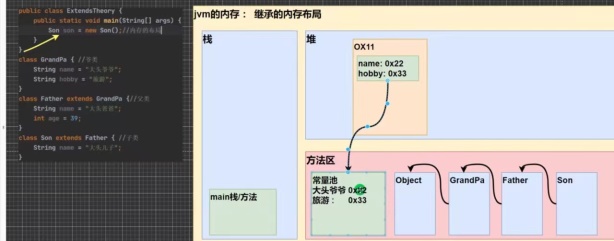
0288 java继承使用细节

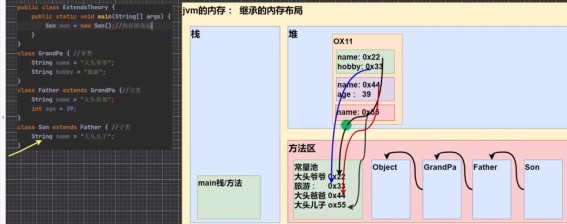
1. 子类继承了所有的属性和方法，非私有属性和方法可以在子类直接访问，但是私有属性和方法不能在子类直接访问，要通过父类提供公共的方法去访问
2. 子类必须调用父类的构造器，完成父类的初始化（创建子类构造器时，父类构造器也被调用）
3. 当创建子类对象时，不管使用子类的哪个构造器，默认情况下总会去调用父类的无参构造器，如果父类没有提供无参构造器，则必须在子类的构造器中用super去指定使用父类的哪个构造器完成对父类的初始化工作，否则，编译不会通过
4. 如果希望指定去调用父类的某个构造器，则显式的调用一下：super(参数列表)
5. super在使用时，必须放在构造器第一行(super只能在构造器中使用)
6. super() 和this()都只能放在构造器第一行，因此这两个方法不能共存在一个构造器
7. java所有类都是Object类的子类
8. 父类构造器的调用不限于直接父类，将一直往上追溯直到Object类（顶级父类）
9. 子类在java中最多只能继承一个父类（指直接继承），即java中是单继承机制
10. 不能滥用继承，子类和父类之前必须满足is-a的逻辑关系

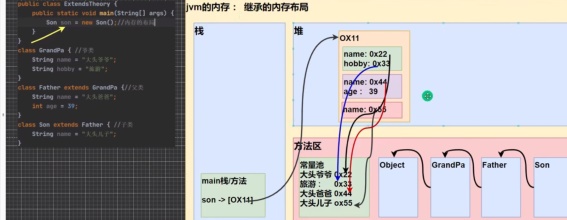
0293 继承本质

继承的内存布局

1. 加载类
2. 在堆里分配空间（按照继承关系）
3. 把对象的地址返回给栈中的对象引用







信息查找规则：

/(1) 先查看子类是否有该属性  
//(2)如果子类有这个属性，并且可以访问，则返回信息  
//(3)如果没有这个属性，就看父类有没有这个属性（如果有该属性，并且可以访问，则返回信息）  
//(4)如果父类没有就按照(3)的规则，继续找上级父类，直到Object

0297 super关键字

基本介绍：

Super代表父类关键字，用于访问父类的属性，方法，构造器

基本属性

1. 访问父类的属性，但不能访问父类的private属性

super.属性名

1. 访问父类的方法，但不能访问父类的private方法

super.方法名(参数列表)

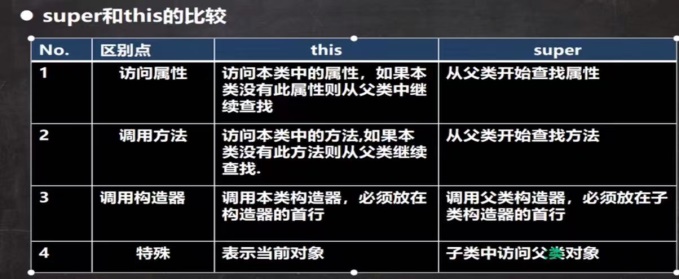
1. 访问父类的构造器：

super(参数列表)；只能放在构造器的第一句，只能出现一句

super关键字的便利/细节

1. 调用父类的构造器的好处：分工明确，父类属性由父类初始化，子类属性由子类初始化
2. 当子类中有和父类中的成员(属性和方法)重名时，为了访问父类的成员，必须通过super。如果没有重名，使用super、this、直接访问是一样的结果
3. Super的访问不限于直接父类，如果爷爷类和本类中有同名的成员，也可以使用super去访问爷爷类的成员，如果多个基类(上级类)中都有同名的成员，使用super访问遵循就近原则。A->B ->C,当然也要遵守访问权限的的相关规则

super和this的比较



0301 方法重写/覆盖(override)//Override

简单的说，方法覆盖(重写)就是子类有一个方法和父类的某个方法的名称、返回类型、参数都一样，那么我们就说子类的这个方法覆盖了父类的方法

注意事项和使用细节：

1. 子类的方法的形参列表、方法名要和父类方法的形参列表，方法名称完全一样
2. 子类方法的返回类型和父类方法的返回类型一样，或者是父类返回类型的子类

比如父类的返回类型是Object，子类的返回类型是String

1. 子类方法不能缩小父类方法的访问权限

public>protected>默认>private

方法重写和重载的比较：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 发生范围 | 方法名 | 形参列表 | 返回类型 | 修饰符 |
| 重载(overload) | 本类 | 必须一样 | 类型、个数或者顺序至少有一个不同 | 无要求 | 无要求 |
| 重写(override) | 父子类 | 必须一样 | 相同 | 子类重写的方法，返回的类型和父类返回的类型一致，或者是其子类 | 子类方法不能缩小父类方法的访问范围 |

0305 面向对象编程-多态

1. 方法的多态

方法或对象具有多种形态，是面向对象的第三大特征，多态是建立在封装的继承基础之上的

方法重载体现多态：传入不同的参数，调用不同的方法，体现多态

方法重写体现多态：对象(类)不一样，调用相应的方法

1. 对象的多态(核心)
2. 一个对象的编译类型和运行类型可以不一致
3. 编译类型在定义对象时，就确定了，不能改变
4. 运行类型是可以变化的
5. 编译类型看定义时 = 号的左边，运行类型看 = 号的右边

让父类的一个引用指向子类的一个对象：

Animal animal = new Dog()；[animal编译类型是Animal，运行类型是Dog]

animal = new Cat();[animal编译类型仍是Animal，运行类型是Cat]

多态注意事项和细节讨论

多态的前提是：两个对象(类)存在继承关系

属性没有重写之说，属性的值看编译类型

instanceOf比较操作符，用于判断对象的运行类型是否为XX类型或XX类型的子类型

多态的向上转型：

1. 本质：父类的引用指向了子类的对象：Animal animal = new Dog();
2. 语法：父类类型 引用名 = new 子类类型();
3. 特点：编译类型看左边，运行类型看右边。可以调用父类中的所有成员（属性，方法）（需遵守访问权限）；**不能调用子类特有的成员方法**
4. 在编译阶段，能调用那些成员，是由编译类型来决定的
5. 最终运行效果看子类的具体实现

编译的时候从父类往下找，运行时从子类往上找

多态的向下转型：

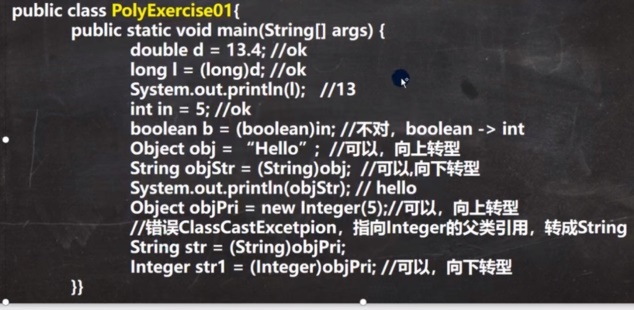
1. 语法：子类类型 引用名 = (子类类型) 父类引用；
2. 只能强转父类的引用，不能强转弗雷德对象
3. 要求父类的引用必须指向的是当前目标类型的对象
4. 当向下转型后，就可以调用子类类型中所有的成员

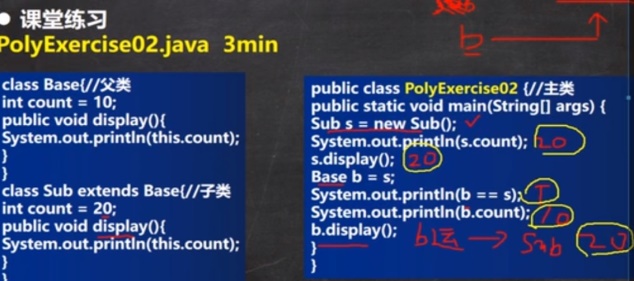
//cat变异类型是Cat，运行类型是Cat

Cat cat = (Cat) animal; **要求animal在向下转型之前就是指向Cat的**

Cat.catchMouse();

练习：





0314 java的动态绑定机制

1. 当调用对象方法的时候，该方法会和该对象的内存地址/运行类型绑定
2. 当调用对象属性时，没有动态绑定机制，哪里声明，哪里使用

多态的应用

1. 多态数组

数组的定义类型为父类类型，里面保存的实际元素类型为子类类型

1. 多态参数

方法定义的形参类型为父类类型，实参类型允许为子类类型

0318 Object类详解

1. equals方法

1）==和equals的对比

（1） ==是一个比较运算符

==既可以判断基本类型，又可以判断引用类型

==如果判断基本类型，判断的是值是否相等

==如果判断引用类型，判断的是地址是否相等，即判定是不是同一个对象

（2）equals

equals是Object类中的方法，只能判断引用类型

默认判断的是地址是否相等，子类中往往重写该方法，用于判断内容是否相等。比如Integer,String

1. hashCode方法：
2. 提高具有哈希结构的容器的效率
3. 两个引用，如果指向的是同一个对象，则哈希值肯定是一样的
4. 两个引用，如果指向的是不同对象，则哈希值是不一样的
5. 哈希值主要是根据地址号来的，不能完全将哈希值等价于地址
6. 后面在集合中，hashCode如果需要的话，也会重写
7. toString方法

基本介绍：

1）默认返回：全类名+@+哈希值的十六进制。子类往往重写toString方法，用于返回对象的属性信息

2）重写toString方法，打印对象或者拼接对象时，都会自动调用该对象的toString形式

3）当直接输出一个对象时，toString方法会被默认的调用：

System.*out*.println(monster);//等价monster.toString方法

1. finalize方法

基本介绍：

1）当对象被回收时，系统自动调用该对象的方法，子类可以重写该方法，做一些释放资源的操作

2）什么时候回收：当某个对象没有任何引用时，则jvm就认为这个对象是一个垃圾对象，就会使用垃圾回收机制来销毁该对象，在销毁对象前，会先调用finalize方法

3）垃圾回收机制的调用，是由系统来决定(即有自己的GC算法)，也可以通过System.gc()主动触发垃圾回收机制

0327 断点调试

1. 在开发中，可以用断点调试，一步一步的看源码执行的过程，从而发现错误所在
2. 在断点调试过程中，是运行状态，是以对象的运行类型来执行的

断点调试是指在程序的某一行设置一个断点，调试时，程序运行到这一行就会停住，然后可一步一步往下调试，调试过程中可以看各个变量当前的值，出错的话，调试到出错的代码行即显示错误，停下。进行分析从而找到这个Bug

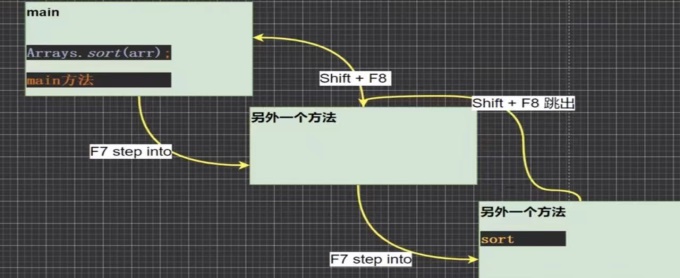
断点调试的快捷键

1. F7(跳入) F8(跳出) F9(resume,执行到下一个断点)

F7：跳入方法内

F8：逐行执行代码

Shift + F8:跳出方法



F9 resume：执行到下一个断点（断点可以在debug过程中，动态的下断点）

项目-------零钱通

项目需求说明：

开发零钱通项目，可以完成收益入账，消费，查看明细，退出系统等功能

项目的界面：

1. 先完成显示菜单，并可以选择
2. 完成零钱通明细
3. 完成收益入账
4. 消费
5. 退出

项目代码实现改进

1. 用户输入4时，给出提示”你确定要退出吗？y/n”,必须输入正确的y/n,否则循环输入指令，直到输入y/n

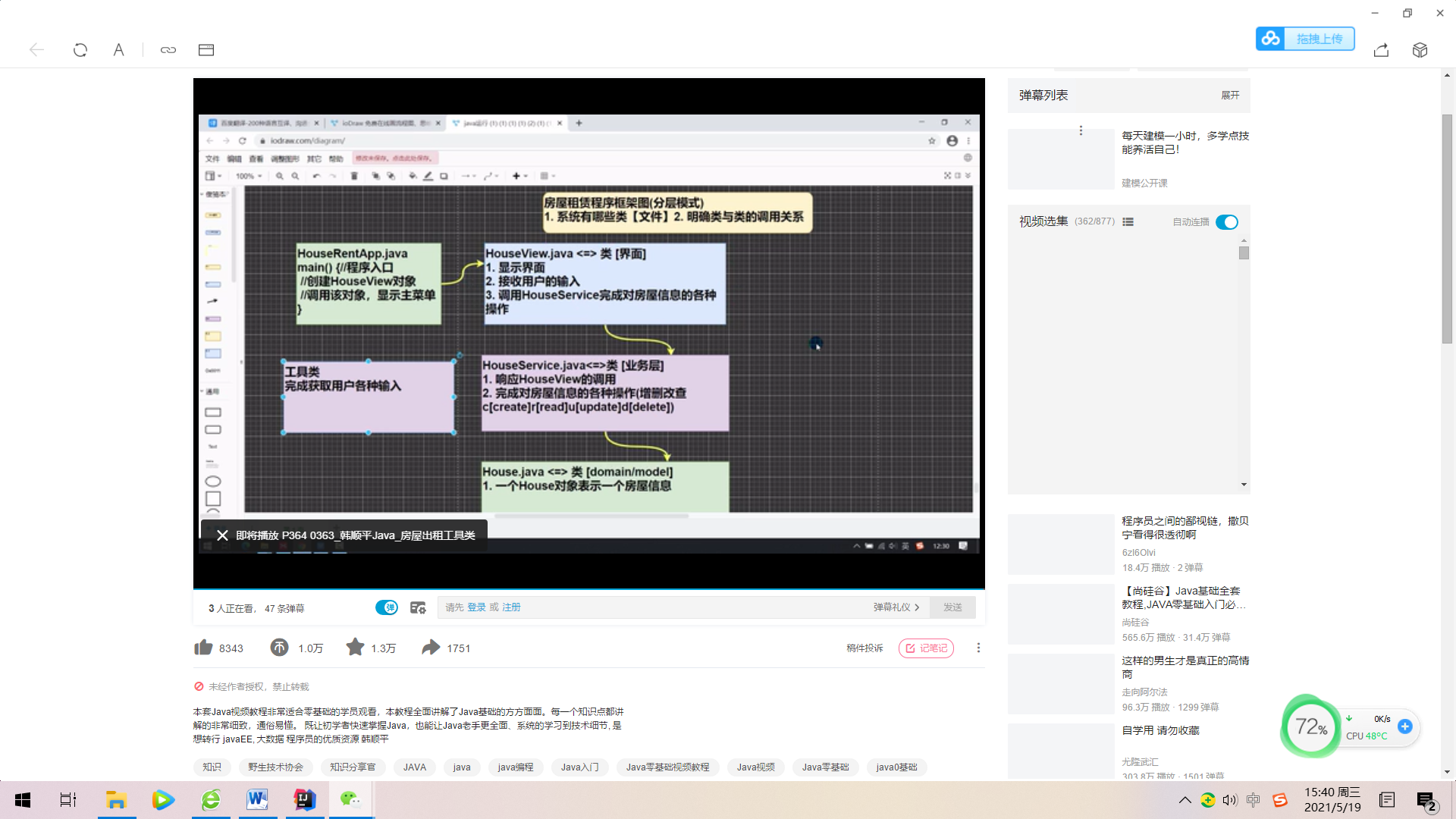
2．在收益入账和消费时，判断金额是否合理，并给出相应提示

3. 将面向过程的代码修改成面向对象的方法

（1）把原来的变量变成属性

（2）把各个功能封装到成员方法里面去，如果再有新功能，封装到方法即可

房屋出租程序项目



项目功能实现：显示主菜单和完成退出软件功能

当一个方法是static时，就是一个静态方法，静态方法可以直接通过类名调用

0373 类变量和类方法

类变量：不同对象共享数据空间

类变量内存剖析：

静态变量被对象共享，因此不影响对静态变量的使用

Jdk7以上版本保存在类对应的Class尾部(堆)

不管static变量在哪里，有以下共识：

1. Static变量是同一个类所有对象共享
2. Static类变量，在类加载的时候就生成了

0374：什么是类变量

类变量也叫静态变量/静态属性，是该类的所有对象共享的变量，任何一个该类的对象去访问它时，取到的都是相同的值，同样任何一个该类的对象去修改它时，修改的也是同一个变量，

类变量使用注意事项和细节

1. 什么时候需要用类变量

当需要让某个类的所有对象都共享一个变量时，就可以考虑使用类变量(静态变量)。

1. 类变量与实例变量(普通属性)区别

类变量是该类的所有对象共享的，而实例变量是每个对象独享的

1. 加上static称为类变量或静态变量，否则称为实例变量/普通变量/非静态成员类名.变量
2. 类变量可以通过类名.类变量名或者对象名.类变量名 来访问，推荐前者。
3. 实例变量不能通过类名.类变量名访 方式访问
4. 类变量在类加载时就初始化了。也就是说，即使没有创建对象，只要类加载了。就可以使用类变量了
5. 类变量的生命周期是随着类的加载开始，随着类消亡而销毁

0378 类方法

类方法基本介绍

类方法也叫静态方法

形式如下：

访问修饰符 static 数据返回类型 方法名() {} 推荐

static 访问修饰符 数据返回类型 方法名() {}

类方法的调用：

使用方式：类名.类方法名 或者 对象名.类方法名

类方法的使用场景：

1. 如果想不创建实例，也可以调用某个方法(即当做工具来使用)，这是把方法做成静态方法非常合适
2. 开发自己的工具类时，可以将方法做成静态的，方便调用
3. 当方法中不涉及到任何和对象相关的成员，则可以将方法设计成静态方法，提高开发效率。

类方法使用注意事项和细节讨论

1. 类方法和普通方法都是随着类的加载而加载，将结构信息存储在方法区；类方法中无this参数，普通方法中隐含着this参数
2. 类方法可以通过类名调用，也可以通过对象名调用
3. 普通方法和对象有关，需要通过对象名调用，比如对象名.方法名(参数)，不能通过类名调用
4. 类方法中不允许使 用和对象有关的关键字，比如this和super。普通方法(成员方法)可以。
5. 类方法(静态方法)只能访问静态变量或方法
6. 普通成员方法，既可以访问非静态成员，也可以访问静态变量(方法)(必须遵守访问权限)

0382 main方法：main是由java虚拟机调用的

main方法的形式public static void main(String[] args){}

1. Java虚拟机需要执行main方法(不是同一个类)，所以该方法的访问权限必须是public.
2. Java虚拟机在执行main()方法时不必创建对象，所以该方法必须是static
3. 该方法接收String类型的数组参数，该数组中保存执行java命令时传递给所运行的类的参数
4. Java执行的程序 参数1 参数2 参数3

[注]：1) 在main()方法中，我们可以直接调用main方法所在类的静态方法或静态属性，但是不能直接访问该类中的非静态成员，必须在创建该类的一个实例对象后，才能通过这个对象去访问类中的非静态成员

0385 代码块

代码块又称为初始化块，属于类中的成员[即是类的一部分] ，类似于方法，将逻辑语句封装在方法体中，通过{}包围起来

但和方法不同，没有方法名，没有返回，没有参数，只有方法体，而且不用通过对象或类显式调用，而是加载类时，或创建对象时隐式调用。

基本语法：

[修饰符] {

代码

}；

说明注意：

1. 修饰符可选，要写的话也只能写static
2. 代码块分为两类，使用static修饰的叫静态代码块，没有static修饰的，叫普通代码块/非静态代码块
3. 逻辑语句可以为任何逻辑语句(输入、输出、方法调用、循环、判断等)
4. ；号可以写上，也可以省略

代码块的好处：

1. 代码块相当于另外一种形式的构造器(对构造器的补充机制)，可以做初始化操作
2. 场景：如果多个构造器中都有重复的语句，就可以抽取到初始化块中，提高代码的重复性

代码块使用的注意事项和细节讨论：

1. static代码块也叫静态代码块，作用就是对类进行初始化，而且它随着类的加载而执行，并且只会执行一次，如果是普通代码块，每创建一个对象，就执行
2. 类什么时候被加载
3. 创建对象实例时(new)
4. 创建子类对象实例，父类也会被加载
5. 使用类的静态成员时(静态属性，静态方法)：代码块先执行，如果有父类，父类的代码块先执行
6. 普通的代码块，在创建对象实例时，会被隐式的调用。被创建一次，就会调用一次

如果只是使用类的静态成员时，普通代码块不会被执行。

1. 创建一个对象时，在一个类调用的顺序是：
2. 调用静态代码块和静态属性初始化(注意：静态代码块和静态属性初始化调用的优先级一样，如果有多个静态代码块和多个静态变量初始化，则按它们定义的顺序调用)
3. 调用普通代码块和普通属性的初始化(注意：普通代码块和普通属性初始化调用的优先级一样，如果有多个普通代码块和多个普通属性初始化，则按定义顺序调用)
4. 调用构造方法
5. 构造器的最前面其实隐含了super()和调用普通代码块。静态相关的代码块，属性初始化，在类加载时，就执行完毕，因此是优先于构造器和普通代码块执行的
6. 创建一个子类对象时(继承关系)，他们的静态代码块，静态属性初始化，普通代码块，普通属性初始化，构造器的调用顺序如下：
7. 父类的静态代码块和静态属性(优先级一样，按定义顺序执行)
8. 子类的静态代码块和静态属性(优先级一样，按定义顺序执行)
9. 父类的普通代码块和普通属性(优先级一样，按定义顺序执行)
10. 父类的构造方法
11. 子类的普通代码块和普通属性(优先级一样，按定义顺序执行)
12. 子类的构造方法
13. 静态代码块只能调用静态成员(静态属性和静态方法)，普通代码块可以调用任意成员

单利设计模式

1. 静态方法和属性的经典使用
2. 设计模式是在大量的实践中总结和理论化之后优选的代码结构、编程风格、以及解决问题的思考方式。

所谓单类设计模式，就是采取一定的方法保证在整个的软件系统中，对某个类只能存在一个对象实例，并且该类值提供一个取得其对象实例的方法

单例模式有两种方式：

1)饿汉式：创建了对象可能没有使用

2)懒汉式：只有使用了才会创建，后面再次调用，会返回上次创建的cat对象

步骤如下：

1) 构造器私有化:防止直接new

2) 类的内部创建对象

3) 向外暴露一个静态的公共方法

饿汉式与懒汉式对比：

1. 二者最主要的区别在于创建对象的时机不同：饿汉式是在类加载就创建了对象实例，而懒汉式是在使用时才创建
2. 饿汉式不存在线程安全问题，懒汉式存在线程安全问题。(后面学习线程会完善)
3. 饿汉式存在浪费资源的可能。因为如果程序员一个对象实例都没有使用，那么饿汉式创建的对象就浪费了，懒汉式是使用时才创建的，就不存在这个问题
4. 在javaSE标准类中，java.lang.Runtime就是经典的单例模式

0393 final关键字

final可以修饰类、属性、方法和局部变量

应用场景：

1. 当不希望类被继承时，可以用final修饰
2. 当不希望父类的某个方法被子类覆盖/重写(override)时，可以用final关键字修饰
3. 当不希望类的某个属性的值被修改，可以用final修饰
4. 当不希望某个局部变量被修改，可以使用final修饰

final使用注意事项和细节讨论

1. final修饰的属性又叫常量，一般用XX\_XX\_XX来命名(不能被修改)
2. final修饰的属性在定义时，必须赋初值。并且以后不能再修改，赋值可以在如下位置之一(选择一个位置赋初值即可)：

1) 定义时

2)在构造器中

3)在代码块中

3. 如果final修饰的属性是静态的，则初始化位置只能是

1)定义时

2)在静态代码块，但不能在构造器中赋值

4. final类不能继承，但是可以实例化对象

5. 如果类不是final类，但是含有final方法，则该方法虽然不能重写，但是可以被继承

6. 一般来说，如果一个类已经是final类，就没有必要再将方法修饰成final方法

7. final不能修饰构造方法即构造器

8. final和static往往搭配使用，效率更高，不会导致类加载，底层编译器做了优化处理

9. 包装类(Integer,Double,Float,Boolean等都是final)，String也是final类

0397 抽象类

当父类的某些方法，需要声明，但是又不确定如何实现时，可以将其声明为抽象方法，用abstract关键字来修饰该方法，用abstract来修饰该类，那么这个类就是抽象类

抽象类的介绍：

1. 用abstract关键字来修饰一个类时，这个类就叫抽象类

访问修饰符 abstract 类名 {

}

1. 用abstract关键字来修饰一个方法时，这个方法就是抽象方法

访问修饰符 abstract 返回类型 方法名(参数列表)；

1. 抽象类的价值更多作用是在于设计，是设计者设计好后，让子类继承并实现抽象类()

抽象类使用的注意事项和细节讨论

1. 抽象类不能实例化
2. 抽象类不一定要包含abstract方法，也就是说，抽象类可以没有abstract方法，但可以有实现方法
3. 一旦类包含了abstract方法，则这个类必须声明为abstract
4. abstract只能修饰类和方法，不能修饰属性和其他的
5. 抽象类可以有任意成员[抽象类本质还是类]，比如：抽象方法顿号构造器、静态属性等等。
6. 抽象方法不能有主体即不能实现
7. 如果一个类继承了抽象类，则它必须实现抽象类的所有抽象方法，除非它自己也声明为abstract类
8. 抽象方法不能使用private、final、 static来修饰，因为这些关键字都是和重写相违背的

0402 接口

接口就是给出一些没有实现的方法，封装到一起，到某个类要使用的时候，再根据具体情况把这些方法写出来，语法：

Interface 接口名 {

//属性

//方法

}

Class 类名 implements 接口{

自己属性；

自己方法；

必须实现的接口的抽象方法

}

小结：

1. 在jdk7.0前，接口里的所有方法都没有方法体，即都是抽象方法
2. Jdk8.0后接口可以有静态方法，默认方法(需要使用default关键字修饰)，也就是说接口中可以有方法的具体实现
3. 在接口中，抽象方法，可以省略abstract关键字
4. 如果一个类implements(实现)接口，需要将该接口的所有抽象方法都实现

接口注意事项和 使用细节：

1. 接口不能被实例化
2. 接口中所有的方法都是public方法，接口中抽象方法，可以不用abstract修饰
3. 一个普通类实现接口，就必须将该接口的所有方法都实现(可使用Alt+Insert)
4. 抽象类实现接口，可以不用实现接口的方法
5. 一个类同时可以实现多个接口
6. 接口的属性中，只能是final的，而且是public static final修饰符。
7. 接口中属性的访问形式：接口名.属性名
8. 接口不能继承其他的类，但是可以继承多个别的接口
9. 接口的修饰符只能是public和默认，这点和类的修饰符是一样的

继承与实现的理解

1. 当子类继承了父类，就自动拥有父类的功能
2. 如果子类需要扩展功能，可以通过实现接口的方式扩展
3. 可以理解实现接口是对java单继承机制的一种补充

继承VS实现

1. 解决的问题不同

继承的价值主要在于：解决代码的重复性和可维护性

接口的价值主要在于：设计，设计好各种规范(方法)，让其它类去实现这些方法，即更加的灵活

1. 接口比继承更加灵活

继承是满足is-a的关系，而接口只需要满足lik e-a的关系

接口在一定程度上实现代码解耦[即：接口规范性+动态绑定]

接口的多态特性：

1. 多态参数：

前面的Usb接口案例，UsbInterface usbinterface,既可以接收手机对象，又可以接收相机对象，体现了接口多态(接口引用可以指向实现了接口的类的对象)

1. 多态数组
2. 接口存在多台传递现象

类的五大成员：

1. 属性
2. 方法
3. 构造器
4. 接口
5. 代码块

0412 内部类

一个类的内部又完整的嵌套了另一个类结构。被嵌套的类称为内部类(inner class),嵌套其他类的类称为外部类(out class)。是我们类的第五大成员

基本语法

class Outer{ //外部类

class Inner{//内部类

}

}

class Other{ //外部其他类

}

内部类的分类

1. 定义在外部类局部位置上(比如方法或代码块内)

1)局部内部类(有类名)：

2)匿名内部类(没有类名)

2. 定义在外部类的成员位置上

1)

2)

0413 局部内部类

局部内部类的使用

说明：局部内部类是定义在外部类的局部位置，比如方法中，并且有类名

1. 可以直接访问外部类的所有成员，包含私有的
2. 不能添加访问修饰符，但是可以用final修饰
3. 作用域：仅仅在定义它的方法或代码块中
4. 局部内部类可以直接访问到外部类的成员
5. 外部类在方法中，可以创建内部类对象，然后调用方法即可(必须在作用域内)
6. 外部其他类不能访问局部内部类(因为局部内部类地位是一个局部变量)
7. 如果外部类和内部类的成员重名时，默认遵循就近原则，如果想访问外部类的成员。使用外部类名.this.成员 去访问

0415 匿名内部类

匿名内部类的使用

1. 本质是类
2. 内部类
3. 该类没有名字
4. 同时还是一个对象

说明：匿名内部类是定义在外部类的局部位置，比如方法中，并且没有类名

匿名内部类的使用：

1. 匿名内部类的基本语法

new 类或接口(参数列表){

类体

}；

1. 匿名内部类既是一个类的定义，同时它本身也是一个对象，因此从语法上看，它既有定义类的特征，也有创建对象的特征，对前面代码分析可以看出这个特点，因此可以调用匿名内部类方法
2. 可以直接访问外部类的所有成员，包含私有的
3. 不能添加访问修饰符，因为它的地位就是一个局部变量
4. 作用域：仅仅在定义它的方法或代码块中
5. 匿名内部类访问外部类成员可直接访问
6. 外部其他类不能访问匿名内部类(因为匿名内部类地位是一个局部变量)
7. 如果外部类和匿名内部类的成员重名时，匿名内部类访问的话，默认遵循就近原则，如果想访问外部类的成员，则可以使用(外部类名.this.成员)去访问

匿名内部类的最佳实践：当做实参直接传递，简洁高效

0419 成员内部类

成员内部类的使用：成员内部类是定义在外部类的成员位置，并且没有static修饰

1. 可以直接访问外部类的所有成员，包含私有的
2. 可以添加任意的访问修饰符(public,protected,默认,private)，因为它的地位就是一个成员
3. 作用域：和外部类的其他成员一样，为整个类体。比如在前面的案例，在外部类的成员方法中创建成员内部类对象，再调用方法
4. 成员内部类访问外部类(比如属性)，直接访问即可
5. 外部类访问成员内部类，创建成员内部类对象，然后使用相关的属性或方法。
6. 外部其他类访问成员内部类：有三种方式

//第一种方式

//outer08.new Inner08():相当于把new Inner08()当做是outer08成员

Outer08.Inner08 inner08 = outer08.new Inner08();

inner08.say();

//第二种方式

//在外部类中编写一个方法，可以返回Inner08对象

Outer08.Inner08 inner08Instance = outer08.getInner08Instance();

inner08Instance.say();

//第三种方式

new Outer08().new Inner08();

1. 如果成员内部类的成员和外部类的成员重名，就会遵守就近原则。可以通过 外部类名.this.属性 来访问外部类的成员

0421 静态内部类

静态内部类的使用：

说明：静态内部类是定义在外部类的成员位置，并且有static修饰

1. 可以直接访问外部类的所有静态成员，包含私有的，但不能直接访问非静态成员
2. 可以添加任意访问修饰符(public,protected,默认,private)，因为它的地位就是一个成员
3. 作用域：同其他的成员，为整个类体
4. 静态内部类访问外部类(比如静态属性)，可以直接访问所有静态成员
5. 外部类访问静态内部类，先创建对象，再访问
6. 外部其他类访问静态内部类：有三种方式

//外部其他类访问静态内部类

//方法一

//因为静态内部类是可以通过类名直接访问访问的，前提是满足访问权限

Outer10.Inner10 inner10 = new Outer10.Inner10();

inner10.say();

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

//方法二:非静态方法

//编写一个方法可以返回静态内部类的实例

Outer10.Inner10 inner101 = outer10.getInner10();

inner101.say();

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

//方法三:静态方法

Outer10.Inner10 inner102 = Outer10.getInner10\_();//不需创建Outer10对象实例

inner102.say();

1. 如果外部类和静态内部类的成员重名时，静态内部类访问时，默认遵循就近原则，如果想访问外部类的成员，则可以使用(外部类名.成员)去访问

枚举：

1. 枚举是一组常量的集合
2. 这里可以理解：枚举属于一种特殊的类，里面只包含一组有限的特定的对象

枚举的二种实现方式

1)自定义类实现枚举

2)使用enum关键字实现枚举

1. 自定义类实现枚举

1)不需要提供setXxx方法，因为枚举对象通常为只读

2)对枚举对象/属性使用final + static共同修饰，实现底层优化

3)枚举对象通常使用全部大写，常量的命名规范

4) 枚举对象根据需要，也可以有多个属性

自定义类实现枚举特点：

1)构造器私有化

2)本类内部创建一个对象

3)对外暴露对象(通过为对象添加public final static修饰符)

4)可以通过get方法，但是不要通过set

1. enum 关键字实现枚举

//1.使用关键字enum代替class

//2.SPRING("春天", "温暖"):常量名(实参列表)。等价于public static final Season SPRING = new Season("春天", "温暖");

//3.如果有多个常量(对象)，使用，间隔即可

//4.如果使用enum来实现枚举，要求将定义的常量对象写在前面

enum 关键字实现枚举注意事项

1. 当我们使enum 关键字开发一个枚举类时，默认会继承Enum类,而且是一个final类
2. 传统的public static final Season SPRING = new Season("春天", "温暖"); 简化成SPRING("春天", "温暖");根据传入的实参可以知道它调用的是哪个构造器
3. 如果使用无参构造器创建枚举对象，则实参列表和小括号可以省略
4. 当有多个枚举对象时，使用，间隔，最后一个分号结尾
5. 枚举对象必须放在枚举类的行首

enum常用方法使用说明

1. toString:Enum类已经重写过了，返回的对象是当前对象名，子类可以重写该方法，用于返回对象的属性信息
2. name:返回当前对象名(常量名)，子类中不能重写
3. ordinal:返回当前枚举类中所有的常量
4. values:返回当前枚举类中所有的常量
5. valueOf:将字符串转换成枚举对象，要求字符串必须为已有的常量名，否则报异常
6. compareTo:比较两个枚举常量(比较的是编号)

结果:return self.ordinal - other.ordinal;

0431 enum实现接口

1. 使用enum关键字后，就不能再继承其他类了，因为enum关键字会隐式继承enum,而java是单继承机制
2. 枚举类的普通类一样，可以实现接口，形式如下：

Enum 类名 implements 接口1，接口2{}

0432 注解

1. 注解(Annotation)也被称为元数据(Metadata)，用于修饰解释包、类、方法、属性、构造器、局部变量等数据信息
2. 和注释一样，注解不影响程序逻辑，但注解可以被编译或运行，相当于嵌入在代码中的补充信息
3. 在javaSE中，注解的使用目的比较简单，例如标记过时的功能，忽略警告等。在javaEE中注解占据了更重要的角色，例如用来配置应用程序的任何切面，代替java EE旧版中遗留的繁冗代码和XML配置等

基本的Annotation介绍

使用Annotation时要在其前面增加@符号，并把Annotation当成一个修饰符使用。用于修饰它支持的程序元素

三个基本的Annotation：

1)@Override:限定某个方法，是重写父类方法，该注解只能用于方法

2)@Deprecated:用于表示某个程序元素(类，方法等)已过时

3)@SuppressWarnings:抑制编译器警告

补充说明：@interface的说明

@interface不是interface，是注解类，是jdk5.0之后加入的

Override使用说明

1. @Override表示指定重写父类的方法(从编译层面验证)，如果父类没有fly方法，则会报错
2. 如果不写@Override注解，而父类仍有public void fly(){},仍然构成重写
3. @Override只能修饰方法，不能修饰其他类，包，属性等等
4. 查看@Override注解原码为@Target(ElementType.METHOD),说明只能修饰方法

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.SOURCE)

public @interface Override {

}

1. @Target是修饰注解的注解，称为元注解

@Deprecated的使用

1. @Deprecated用于表示某个程序元素(类、方法等)，已过时
2. 可以用于修饰方法、类字段、包、参数等等
3. @Target(value={CONSTRUCTOR, FIELD(字段，属性), LOCAL\_VARIABLE, METHOD, PACKAGE, PARAMETER, TYPE})
4. @Deprecated的作用可以做到新旧版本的兼容和过渡

@SuppressWarnings的使用

1.当不希望看到这些警告的时候，可以使用SuppressWarnings注解来抑制警告信息

2.在{""}中，可以写入所希望抑制(不显示)警告信息

3.可以指定的警告类型有

4.关于@SuppressWarnings的作用范围，和放置的位置有关，比如把@SuppressWarnings放在main方法里,那么作用范围就是main方法，通常可以放置在具体的语句，方法，类上

5.@SuppressWarnings源码

5.1放置位置是TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL\_VARIABLE

5.2该注解类有数组String[] value(),可以设置数组

/\*

@Target({TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL\_VARIABLE})

@Retention(RetentionPolicy.SOURCE)

public @interface SuppressWarnings {

String[] value();

}

\*/

6.说明各种值：

1)unchecked是忽略没有检查的警告

2)rawtypes是忽略没有指定泛型的警告(传参时没有指定泛型的警告错误)

3)unused是忽略没有使用某个变量的警告错误

4) @SuppressWarnings可以修饰的程序元素为{TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL\_VARIABLE}

5)生成@SuppressWarnings时，直接点击右侧的黄色提示，就可以选择(注意可以指定生成的位置)

JDK的元Annotation

1. 元注解的基本介绍

JDK的元Annotation用于修饰其他Annotation

1. 元注解的种类

1)Retention//指定注解的作用范围，三种SOURCE,CLASS,RUNTIME

2)Target//指定注解可以在哪些地方使用

3)Document//指定该注解是否会在javacode体现

4)Inherited//子类会继承父类注解

1. Retention注解：

只能用于修饰一个Annotation定义，用于指定该Annotation可以保留多长时间，@ Retention包含一个RetentionPolicy类型的成员变量，使用@ Retention时必须为该value成员变量指定值

@ Retention的三种值

1) RetentionPolicy. SOURCE:编译器使用后，直接丢弃这种策略的注解

2) RetentionPolicy. CLASS：编译器将把注解记录在class文件中，当运行Java程序时，JVM不会保留注解，这是默认值

3) RetentionPolicy. RUNTIME：编译器将把注解记录在class文件中，当运行java程序时，JVM会保留注解，程序可以通过反射获取该注解

1. Target注解

用于修饰Annotation定义,用于指定被修饰的Annotation能用于修饰那些程序元素， @Target也包含一个名为value的成员变量

1. Document注解

用于指定被改元Annotation修饰的Annotation类将被javadoc工具提取成文档，即在生成文档时，可以看到该注解

说明：定义为Documented的注解必须设置Retention为RUNTIME

1. Inherited

被它修饰的Annotation将具有继承性，如果某个类使用了被@Inherited修饰的Annotation，则其子类将自动具有该注解