笔记总链接：http://bbs.itheima.com/thread-200600-1-1.html

Chapter 1 Java概述

一、Java语言基础组成-Part 1

Java语言基础由关键字、标识符、注释、常量和变量、运算符、语句、函数和数组等组成。

1.1 关键字

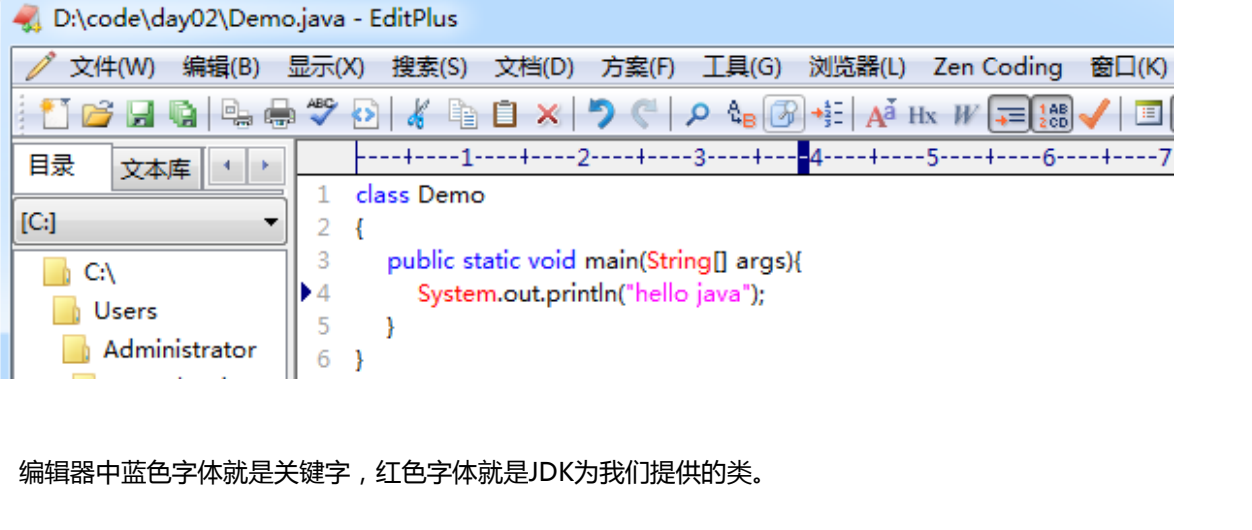
关键字的定义和特点：

定义：被Java语言赋予了特殊含义的单词。

特点：关键字中所有字母都为小写。

关键字的定义和特点				
定义：被Java语言赋予了特殊含义的单词 特点：关键字中所有字母都为小写				
用于定义数据类型的关键字				
class	interface	byte	short	int
long	float	double	char	boolean
void				
用于定义数据类型值的关键字				
true	false	null		
用于定义流程控制的关键字				
if	else	switch	case	default
while	do	for	break	continue
return				
用于定义访问权限修饰符的关键字				
private	protected	public		
用于定义类，函数，变量修饰符的关键字				
abstract	final	static	synchronized	
用于定义类与类之间关系的关键字				
extends	implements			
用于定义建立实例及引用实例，判断实例的关键字				
new	this	super	instanceof	
用于异常处理的关键字				
try	catch	finally	throw	throws
用于包的关键字				
package	import			
其他修饰符关键字				
native	strictfp	transient	volatile	assert

示例：使用编辑器EditPlus编写一个Hello World输出程序。



编辑器中蓝色字体就是关键字，红色字体就是JDK为我们提供的类。

P.S.

1、所有的关键字都不需要记忆，在后期的学习中慢慢就会掌握。

2、EditPlus默认会创建一个后缀名为bak的备份文件，如果不想创建备份文件，只需要通过如下操作：点击Tools-->preferences-->File-->勾掉Create backup file when saving即可。

3、类名的首字母要大写，这是Java语言的命名规范之一。

4、写代码一定要遵守代码规范，注重代码的阅读性。

5、起名字一定要见明知意，例如，Abc这种名称就不要写，DemoTest则可以。

1.2 标识符

1.2.1 定义及特点

定义：在程序中自定义的一些名称，例如：类名。

特点：由26个英文字母大小写，数字：0-9，符号：_、$组成。

1.2.2 定义合法标识符规则

1. 数字不可以开头，例如：2Demo就不可以。
2. 不可以使用关键字，例如：public就不可以，因为public是关键字。
3. 不可以包含“_、$”以外的特殊符号，例如：“Demo Test”就不可以，其中的空格就是特殊符号。

P.S.

1、Java中严格区分大小写。

2、在起名字时，为了提高阅读性，要尽量有意义。

3、公司中经常会通过“_”代表某一类名称，例如：_temp。

通过“$”分隔主名称与子名称，例如：TrafficLamp$Red。

1.2.3 Java中的名称规范

包名：多单词组成时所有字母都小写。

例如：xxxyyyzzz

类名接口名：多单词组成时，所有单词的首字母大写。

例如：XxxYyyZzz

变量名和函数名：多单词组成时，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。

例如：xxxYyyZzz

常量名：所有字母都大写。多单词时每个单词用下划线连接。

例如：XXX_YYY_ZZZ

1.3 注释

1.3.1 定义及特点

定义：用于注解说明解释程序的文字就是注释。

特点：提高了代码的阅读性。

1.3.2 Java中的注释格式

1. 单行注释

格式：//注释文字

2. 多行注释

格式：/* 注释文字 */

3. 文档注释

格式：/** 注释文字 */

示例：

```
/**
这是我的Hello World程序。
@author 小强
*/
class Demo
{
    /*
    这是主函数，是程序的入口。
    它的出现可以保证程序的独立运行。
    */
    public static void main(String[] args)
    {
        //这是输出语句用于将括号内的数据打印到控制台。
        System.out.println("Hello World");
    }
}
```

复制代码

P.S.

1、对于单行和多行注释，被注释的文字，不会被JVM（java虚拟机）解释执行。所以，即使添加再多的注释，编译后生成的class文件占用硬盘字节多少不变。

2、对于文档注释，是java特有的注释，其中注释内容可以被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。

3、注释是一个程序员必须要具有的良好编程习惯。初学者编写程序必须养成习惯：先写注释再写代码。

4、将自己的思想通过注释先整理出来，再用代码去体现，因为代码仅仅是思想的一种体现形式而已。

示例：

```
/*
需求：练习一个hello world程序。

思路：
1. 定义一个类，因为java程序都定义在类中，java程序都是以类的形式形成的，类的形式其实就是一个字节码文件的最终体现。
2. 定义一个主函数，为了让该类可以独立运行。
3. 因为要演示hello world，在控制台上看到该字样，所以需要使用输出语句完成。

步骤：
1. 用class关键字来完成类的定义，并起一个阅读性强的类名。
2. 主函数：public static void main(String[] args);，这是固定格式的，jvm认识。
3. 使用输出语句：System.out.println("hello world");。
*/
class Demo
{
   //定义一个主函数，为了保证程序的独立运行。
   public static void main(String[] args){
      //这是输出语句，用于将括号中的数据打印到控制台上，ln可以在数据的结尾处换行。
      System.out.println("hello world");
   }
}
```

复制代码

5、单行注释可以嵌套单行注释，单行注释可以嵌套多行注释，多行注释可以嵌套单行注释。但是，多行注释不能嵌套多行注释，因为多行注释的开头会和被嵌套的多行注释的结尾配对，导致后面的注释失效。

6、可以使用注释对代码中的错误进行定位。

方法：当程序运行报错时，将某些代码注释掉，然后重新编译，运行。如果程序不再报错，那么说明注释掉的部分代码中包含错误代码。

1.4 常量

1.4.1 定义

常量表示不能改变的数值。

1.4.2 Java中常量的分类

1. 整数常量：所有整数。
2. 小数常量：所有小数。
3. 布尔(boolean)型常量：只有两个数值，true、false。
4. 字符常量：将一个数字字母或者符号用单引号(' ')标识，如：'a'。
5. 字符串常量：将一个或者多个字符用双引号("")标识，如："hello world"、"a"、""（空字符串）。
6. null常量：只有一个数值就是：null。

1.4.3 进制的由来

对于整数，有四种表现形式：

二进制：0-1，满2进1。

八进制：0-7，满8进1，用0开头表示，如：012。

十进制：0-9，满10进1。

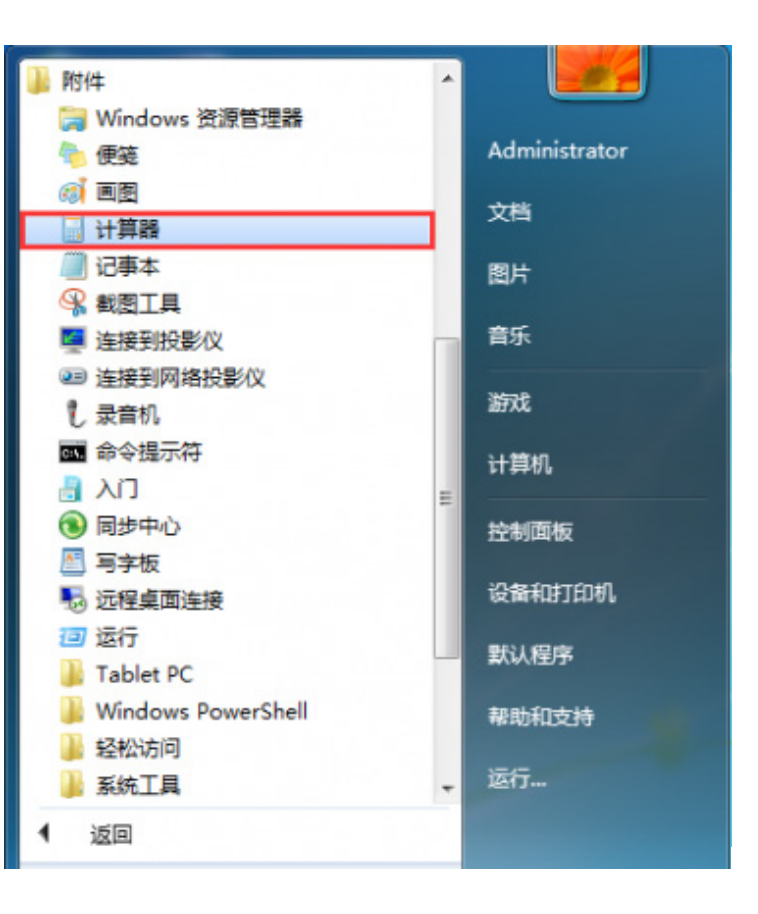
十六进制：0-9，A-F，满16进1，用0x开头表示。如：0x4A2C。

任何数据在计算机中都是以二进制的形式存在的，二进制早期由电信号开关演变而来。一个整数在内存中一样也是二进制的，但是使用一大串的1或者0组成的数值进行使用很麻烦。所以就想把一大串缩短点，将二进制中的三位用一位表示。这三位可以取到的最大值就是7，超过7就进位了，这就是八进制。但是对于过长的二进制变成八进制还是较长，所以出现的用4个二进制位表示一位的情况，四个二进制位最大是15，这就是十六进制。

规律：进制越大，表现形式越短。

示例：使用计算器进行进制转换。

1. 点击“开始”-->“所有程序”-->“附件”-->“计算器”，点击“查看”-->“程序员”。



2. 在十进制下，输入1100。

3. 点击二进制，结果如下：

4. 点击八进制，结果如下：

5. 点击十六进制，结果如下：

P.S.

1Byte = 8bit

1KiB = 1024Byte

1MiB = 1024KiB

1GiB = 1024MiB

1.4.4 进制的基本转换

1. 十进制转二进制。

原理：对十进制数进行除2运算。

示例：求十进制数6的二进制数。

6/2 = 3 余0

3/2 = 1 余1

1/2 = 0 余1

故，6（十进制）= 110（二进制）。

2. 二进制转十进制。

原理：二进制乘以2的过程。

示例：求二进制数110的十进制数。

110 = 0*2（0）+1*2（1）+1*2（2）= 0 + 2 + 4 = 6

附：括号中的数值代表次方。

示例：求二进制数00110101的十进制数。

```
     0   0   1   1   0   1   0   1
   128  64  32  16   8   4   2   1
------------------------------
= 32 * 1 + 16 * 1 + 4 * 1 + 1 * 1 = 53
```

示例：二进制加法，5 + 4 = 9。

```
   1 0 1
+  1 0 0
------------
 1 0 0 1
```

3. 十进制转八进制。

原理：八进制，其实就是二进制位，3个二进制位，1个八进制位。

示例：求十进制数43的八进制数。

十进制　43

二进制　101011

三位分割 000-101-011

八进制　　0　5　3

因此，43（十进制）= 101011（二进制）= 053（八进制）。

4. 十进制转十六进制。

原理：十六进制，其实就是二进制位，4个二进制位，1个十六进制位。

示例：求十进制数43的十六进制数。

十进制　43

二进制　101011

四位分割 0010-1011

十六进制 2(2) 11(B)

因此，43（十进制）= 101011（二进制）= 0x2B（十六进制）

1.4.5 负数的进制

原理：负数的二进制表现形式就是对应的正数二进制取反加1。

示例：求-6的二进制表现形式，其实就是对应的正数6的二进制取反加1。

```
               6
      0000-0110
取反  1111-1001
加1  +0000-0001
-------------------
      1111-1010
```

P.S.

负数的二进制最高位永远是1。

1.5 变量

1.5.1 变量的概念

定义：内存中的一个存储区域，该区域有自己的名称（变量名）和类型（数据类型），该区域的数据可以在同一类型范围内不断变化。

特点：变量其实就是将不确定的数据进行存储，也就是需要在内存中开辟一个空间。

为什么要定义变量？

用来不断的存放同一类型的常量，并可以重复使用。

定义变量的格式：

数据类型　变量名 = 初始化值；

例如：byte b = 3;

P.S.

1、格式是固定的，记住格式，以不变应万变。

2、变量的作用范围（一对{}之间有效）。

3、变量只能存放某一类型的数据。

理解：

变量就如同数学中的未知数X。

1.5.2 Java语言的数据类型

Java语言是强类型语言，对于每一种数据都定义了明确的具体数据类型，在内存中分配了不同大小的内存空间：

byte　1个字节

short　2个字节

int　4个字节

long　8个字节

float　4个字节

double 8个字节

char　2个字节

Java语言的数据类型包括8种基本类型，3种引用类型。

P.S.

1、整数默认类型：int类型，小数默认类型：double类型。

2、double类型的小数精度比float类型的小数更高。

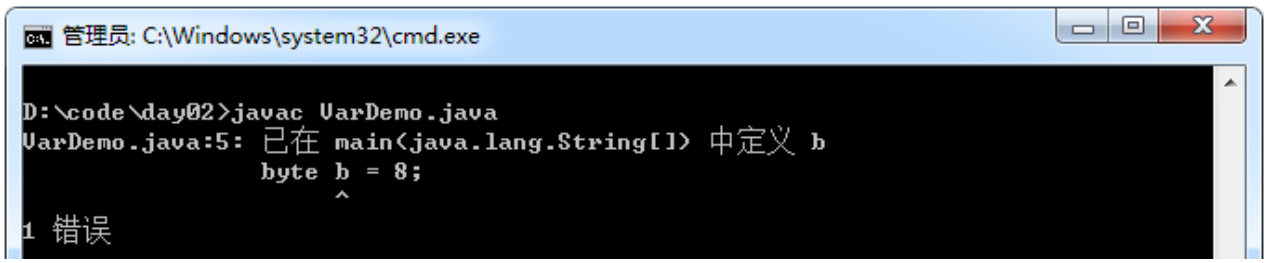
1.5.3 常见错误

错误示例一：

```
class VarDemo
{
    public static void main(String[] args){
        byte b = 3;
        byte b = 8;
    }
}
```

复制代码

运行结果：



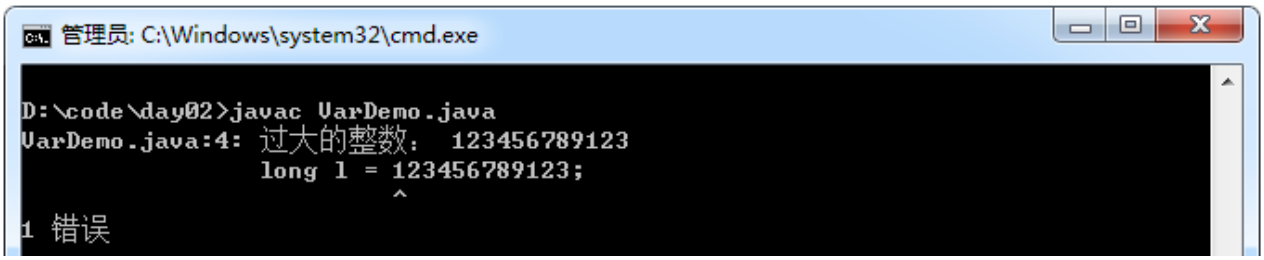
错误原因：变量定义一次以后，不能再重新定义一次。

错误示例二：

```
class VarDemo
{
    public static void main(String[] args){
        long l = 123456789123;
        System.out.println(l);
    }
}
```

复制代码

运行结果：



错误原因：由于整数默认类型是int类型，如果数值超过了int类型的范围，那么就会报如上错误。即使是赋值给long类型的变量，但是由于后面的常量已经超过了int类型的范围，同样会报错。

解决方法：在数值后面加上一个“l”，就可以让编译器知道后面的常量是long类型。

代码如下：

```
class VarDemo
{
    public static void main(String[] args){
        long l = 123456789123l;
        System.out.println(l);
    }
}
```

复制代码

运行结果：

错误示例三：

```
01.  class VarDemo
02.  {
03.      public static void main(String[] args){
04.          float f = 2.3;
05.          System.out.println(f);
06.      }
07.  }
```

复制代码

错误原因：由于小数默认是double（8byte）类型，赋值给float（4byte）类型的变量，当然可能会损失精度，必然通不过编译器的审核。

解决方法：在数值后面加上一个“f”，让编译器知道后面的常量是float类型的。

```
01.  class VarDemo
02.  {
03.      public static void main(String[] args){
04.          float f = 2.3f;
05.          System.out.println(f);
06.      }
07.  }
```

复制代码

运行结果：

错误示例四：

```
01.  class VarDemo
02.  {
03.      public static void main(String[] args){
04.          int y;
05.          System.out.println(y);
06.      }
07.  }
```

复制代码

运行结果：

```
管理员: C:\Windows\system32\cmd.exe

D:\code\day02>javac VarDemo.java
VarDemo.java:5: 可能尚未初始化变量 y
                System.out.println(y);
                                   ^
1 错误
```

错误原因：由于变量y是未初始化的，所以不能直接操作。

解决方法：一定要为变量y先赋值，再打印输出。

错误示例五：

```
class VarDemo
{
    public static void main(String[] args){
        int z = 9;
    }
    System.out.println(z);
}
```

复制代码

运行结果：

```
管理员: C:\Windows\system32\cmd.exe

D:\code\day02>javac VarDemo.java
VarDemo.java:6: 需要 <标识符>
        System.out.println(z);
                           ^
VarDemo.java:6: 需要 <标识符>
        System.out.println(z);
                           ^
2 错误
```

错误原因：由于变量z的作用范围是在一对{}之间有效，超出这个范围就失效了。所以，找不到z这个符号。

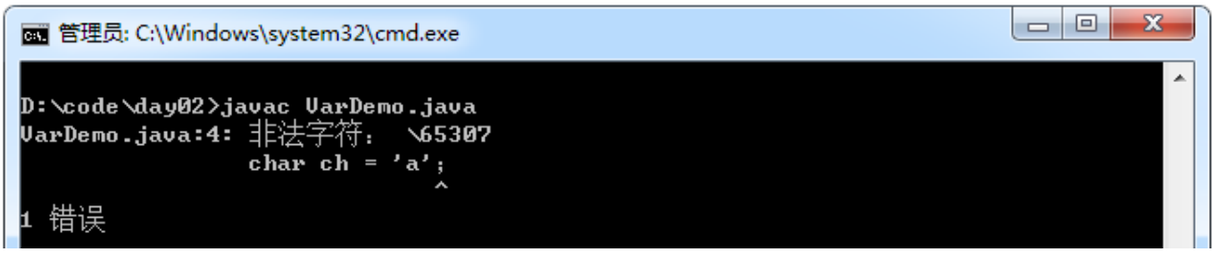
解决方法：将大括号去掉，或者在大括号里面直接打印变量z。

错误示例六：

```
class VarDemo
{
    public static void main(String[] args){
        char ch = 'a';
        System.out.println(ch);
    }
}
```

复制代码

运行结果：



错误原因：编写程序时，不能使用中文符号。

解决方法：改成英文分号符号即可。

1.5.4 类型转换

类型转换在开发中也很常用，简单来说就是类型之间相互的转化，类型转换共分两种，自动类型转换和强制类型转换。

1. 自动类型转换（隐式类型转换）

定义：自动类型转换就是不需要我们手动对类型来进行强制转换。

示例：

```
class VarDemo
{
    public static void main(String[] args){
        int x = 3;
        byte b = 5;
        x = x + b;
        System.out.println(x);
    }
}
```

复制代码

运行结果：

说明：int类型的变量占4个字节，当byte类型的变量与其相加的时候，首先会将byte类型的变量自动转化为4个字节的int类型，然后再进行加法操作。

2. 强制类型转换（显式类型转换）

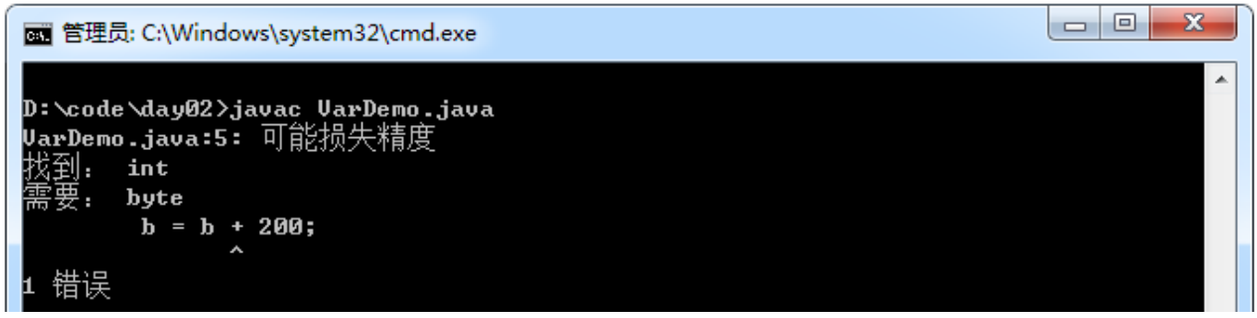
定义：强制类型转换需要把类型进行手动转换，否则无法正常使用

示例：

```
class VarDemo
{
    public static void main(String[] args){
        byte b = 3;
        b = b + 200;
        System.out.println(b);
    }
}
```

复制代码

运行结果：



错误原因：当byte类型的变量提升为int类型与int类型的常量200相加后，结果依然是int类型，再赋值给byte类型，当然会出现损失精度的错误。

解决方法：进行强制类型转换，也就是将占4个字节的int类型值，再强硬存储到占1个字节的byte变量中。

代码如下：

```
class VarDemo
{
    public static void main(String[] args){
        byte b = 3;
        b = (byte)(b + 200);
        System.out.println(b);
    }
}
```

复制代码

运行结果：

```
管理员: C:\Windows\system32\cmd.exe
D:\code\day02>javac VarDemo.java

D:\code\day02>java VarDemo
-53
```

说明：结果为-53的原因是，200+3结果是203，换算成占1个字节空间的二进制也就是11001011。由于首位为1，所以是负数，除符号位外，其余部分取反加1，得1,0110101，即-53。

P.S.

1、只有数值类型才能进行加法操作，非数值类型不行。

示例：

```
class VarDemo
{
    public static void main(String[] args){
        System.out.println(true+1);
    }
}
```

复制代码

运行结果：

```
管理员: C:\Windows\system32\cmd.exe
D:\code\day02>javac VarDemo.java
VarDemo.java:4: 运算符 + 不能应用于 boolean,int
        System.out.println(true+1);
                           ^
1 错误
```

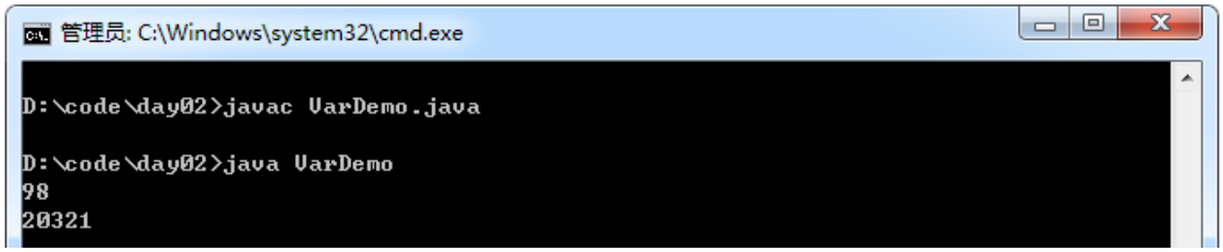
2、char类型数据也可以和int类型相加，但是首先char类型数据会被自动提升为int类型。

示例：

```
class VarDemo
{
    public static void main(String[] args){
        System.out.println('a'+1);
        System.out.println('你'+1);
    }
}
```

复制代码

运行结果：



说明：字符类型数据之所以能够自动提升为int类型是因为字符类型数据在计算机中也是用0、1表示的，int类型数据在计算机中也用0、1表示，所以char类型数据当然可以转换为int类型数据。但是，字符类型数据在计算机中使用0、1表示是按照何种顺序组合排列的则需要依据某个码表而定。Java中的内置码表是Unicode，既包含中文，也包含英文。

P.S.

通过强转也可以把数字强转成字符。

示例：

```
class VarDemo
{
    public static void main(String[] args){
        System.out.println((char)('a'+1));
    }
}
```

复制代码

运行结果：

表达式的数据类型自动提升：

所有的byte型、short型和char的值将被提升到int型。

如果一个操作数是long型，计算结果就是long型；

如果一个操作数是float型，计算结果就是float型；

如果一个操作数是double型，计算结果就是double型。

面试难题：

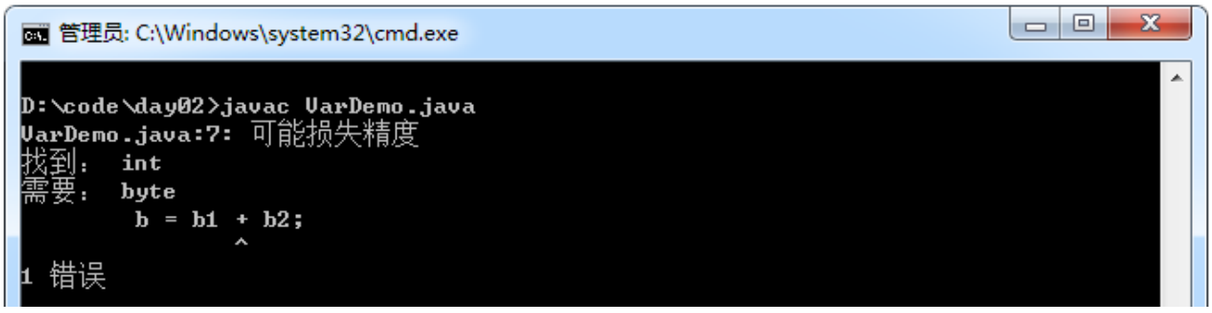
说出报出如下错误的原因：

```
class VarDemo
{
    public static void main(String[] args){
        byte b = 3 + 7;
        byte b1 = 3;
        byte b2 = 7;
        b = b1 + b2;

        System.out.println(b)
    }
}
```

复制代码

运行结果：



错误原因：涉及到编译器编译程序时候的细节，之所以byte b = 3 +7;，没有报错，是因为3和7都是常量，编译器知道结果是10，并且在byte范围之内，因此就自动进行了强转，所以不会报错。而b = b1 + b2;中b1和b2都是变量，编译器编译程序是一行一行编译的，它根本不知道b1和b2到底是多少，两个byte类型的数据相加时，首先都会被提升为int类型，他们的和也是int类型，其值可能会超过byte的范围，因此就会报错。

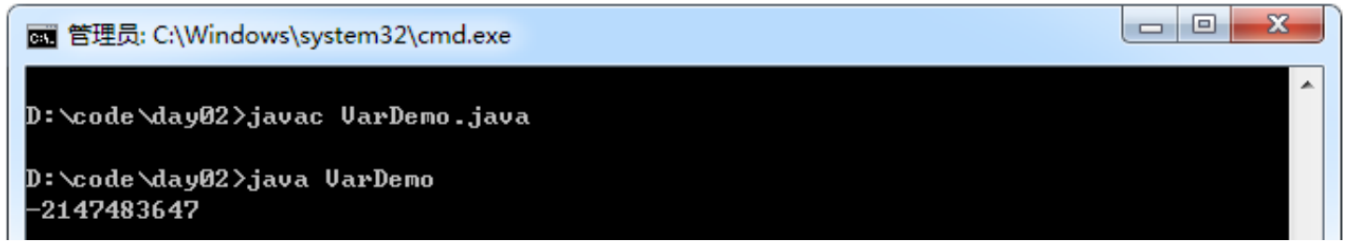
而如下程序则不会报错：

```
class VarDemo
{
    public static void main(String[] args){
        int x;
        int x1 = Integer.MAX_VALUE;
        int x2 = 2;
        x = x1 + x2;

        System.out.println(x);
    }
}
```

复制代码

运行结果：



说明：原因是因为int类型的两个变量相加，最后还是int类型，虽然结果溢出，但是不会报错。

1.6 运算符

1.6.1 算数运算符

算术运算符

运算符	运算	范例	结果
+	正号	+3	3
-	负号	b=4;-b;	-4
+	加	5+5	10
-	减	6-4	2
*	乘	3*4	12
/	除	5/5	1
%	取模	5%5	0
++	自增（前）	a=2;b=++a;	a=3;b=3
++	自增（后）	a=2;b=a++;	a=3;b=2
--	自减（前）	a=2;b=--a	a=1;b=1
--	自减（后）	a=2;b=a--	a=1;b=2
+	字符串相加	"He"+"llo"	"Hello"

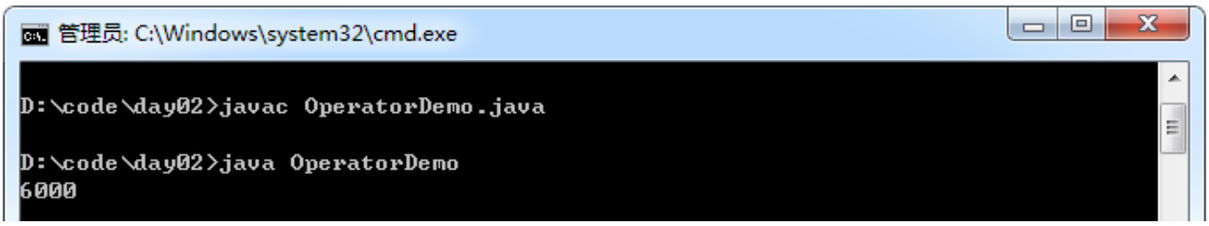
示例1：

```
class OperatorDemo
{
    public static void main(String[] args){
        int x = 6370;
```

```
        x = x / 1000 * 1000;
        System.out.println(x);
    }
}
```

复制代码

运行结果：



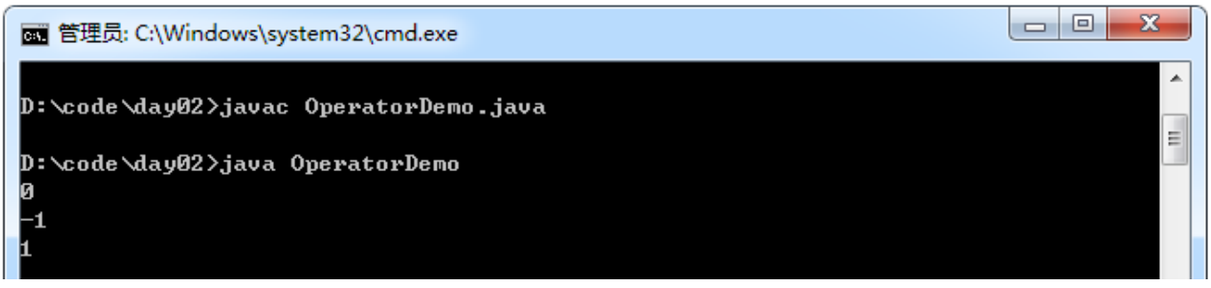
说明：整数与整数相除时，结果永远是整数，小数部分被忽略。

示例2：

```
class OperatorDemo
{
    public static void main(String[] args){
        System.out.println(5%5);
        System.out.println(-5%2);
        System.out.println(5%-2);
    }
}
```

复制代码

运行结果：



说明：

负数对正数取模结果为负数。

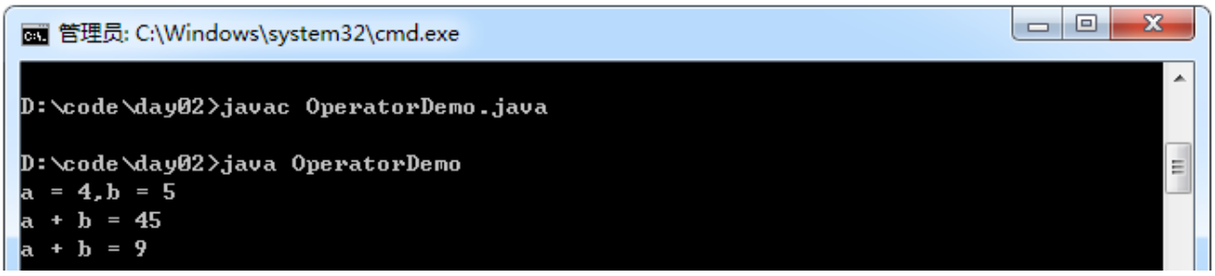
正数对负数取模结果为正数。

示例3：

```
class OperatorDemo
{
    public static void main(String[] args){
        int a = 4,b = 5;
        System.out.println("a = " + a + ",b = " + b);
        System.out.println("a + b = " + a + b);
        System.out.println("a + b = " + (a + b));
    }
}
```

复制代码

运行结果：



P.S.

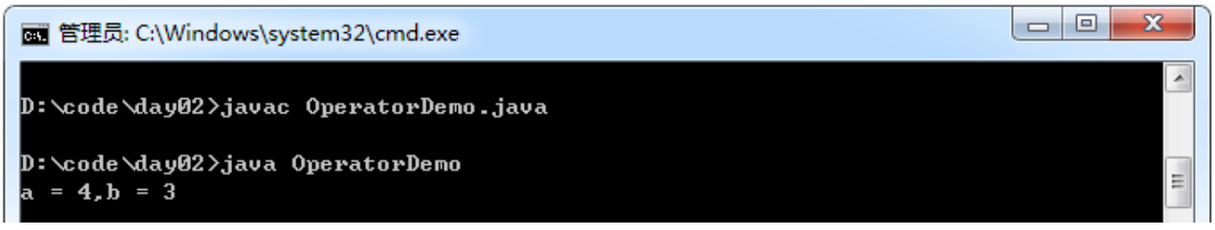
加号也可以作为连接符使用。

示例4：

```
class OperatorDemo
{
    public static void main(String[] args){
        int a = 3,b;
        b = a++;
        System.out.println("a = " + a + ",b = " + b);
    }
}
```

复制代码

运行结果：



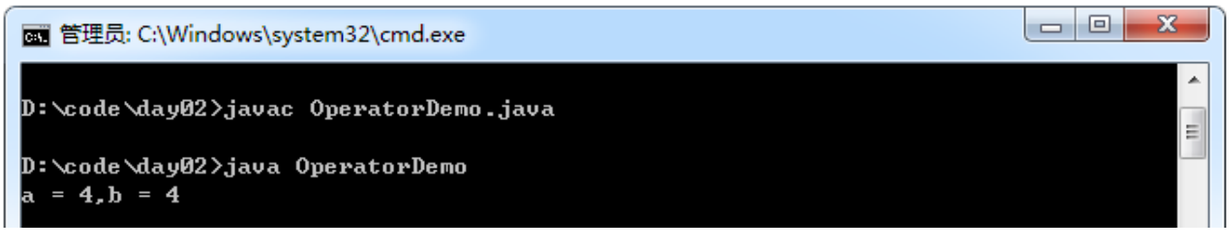
说明：计算机中的实际操作为：当执行b = a++;语句时，先把a放在一个临时内存空间中，然后将a自加1，再将临时内存空间中的a赋值给b，因此b还是原来的a的值，也就是3。

示例5：

```
class OperatorDemo
{
    public static void main(String[] args){
        int a = 3,b;
        b = ++a;
        System.out.println("a = " + a + ",b = " + b);
    }
}
```

复制代码

运行结果：



说明：当执行b = ++a;语句时，先将a自加1，a此时的值为4，再将a赋值给b，因此b的值也是4。

1.6.2 赋值运算符

符号：= , +=, -=, *=, /=, %=

示例1：加号和赋值符号是可以合在一起使用的。

```
class OperatorDemo
{
    public static void main(String[] args){
        int a = 4;
        a += 2;
        System.out.println("a = " + a);
    }
}
```

复制代码

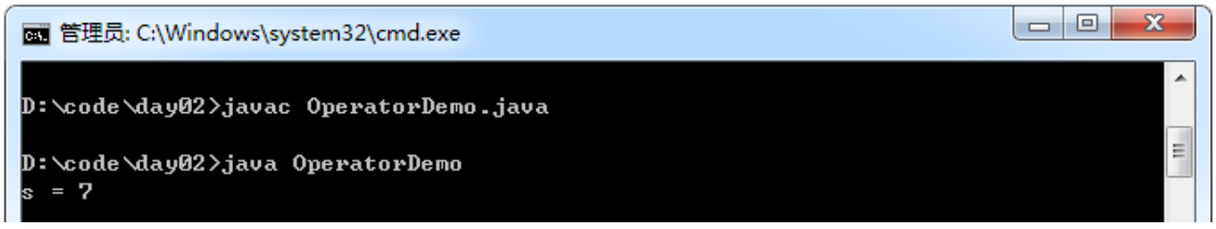
运行结果：

示例2：比较s += 4;和s = s + 4;的不同。

```
class OperatorDemo
{
    public static void main(String[] args){
        short s = 3;
        s += 4;
        System.out.println("s = " + s);
    }
}
```

复制代码

运行结果：



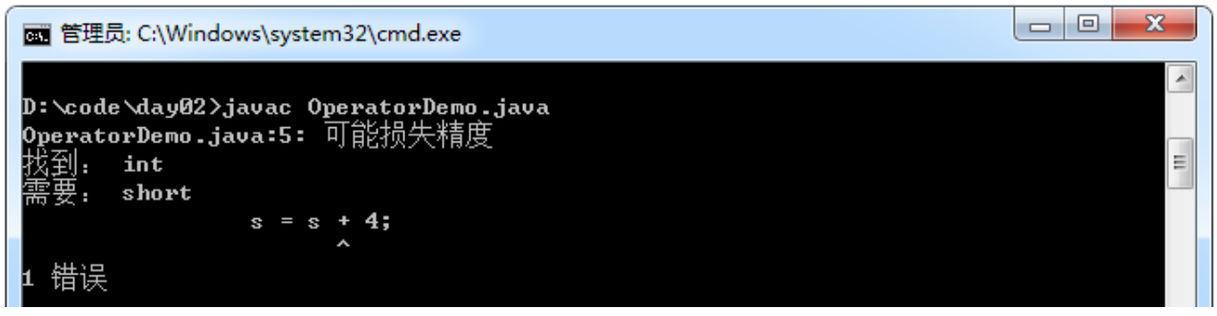
说明：在执行s+=4;语句时，编译器在编译的时候，默认进行了强制类型转换，也就是将int类型的数据转换成short类型的数据。

示例3：

```
class OperatorDemo
{
    public static void main(String[] args){
        short s = 3;
        s = s + 4;
        System.out.println("s = " + s);
    }
}
```

复制代码

运算结果：



说明：在执行s = s + 4;语句时，编译器在编译的时候，默认并没有强制类型转换。所以，s是short类型，4是int类型，s会自动提升为int类型，相加的和也是int类型，赋值给short类型的变量肯定会损失精度。这时候就需要进行强制类型转换：s = (short)(s + 4);。

~END~

~爱上海，爱黑马~