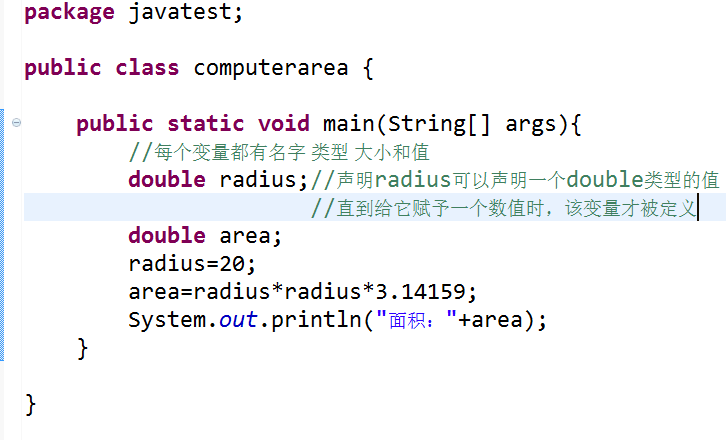
# 第二章 基本程序设计

## 2.1引言

如何编程解决实际问题。

## 2.2编写简单的程序

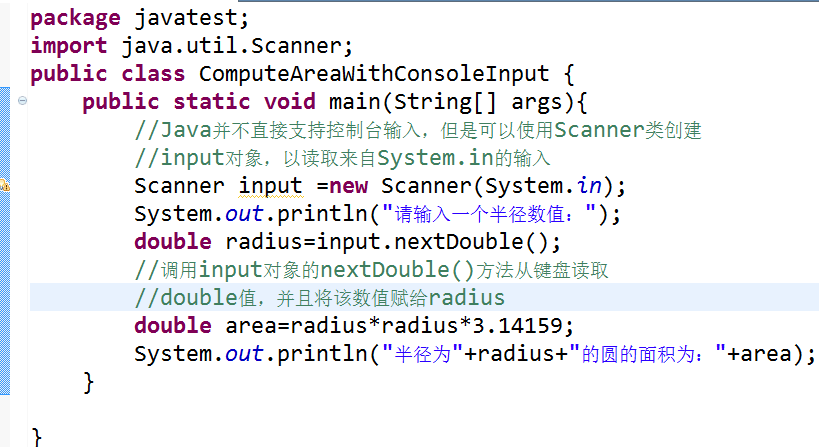


1. 加号（+）有两种意义：一种用途是做加法；另一种用途是做字符串的连接（合并）
2. 警告：在源代码中，字符串量不能跨行。

## 2.3从控制台读取输入

1. 要点提示：从控制台读取输入，使得程序可以从用户那里获得输入。

2. 可以使用Scanner类从控制台输入。



## 2.4标识符

1. 要点提示：标识符是为了标志程序中诸如类，方法和变量的元素而采用的命名。

2. 标识符是用于命名程序中的变量，方法，类和其它项。具有描述性的标识符可以提高程序的可读性。避免采用缩写作为标识符。

## 2.5变量

1. 要点提示：变量用于表示程序中可能被改变的值。

2. 在赋值给变量之前，必须声明变量。方法中声明的变量在使用之前必须被赋值。任何时候，都要尽可能一步完成变量的声明和赋初值。这会使程序易读，同时避免设计错误。

3. 每个变量都有适用范围。

## 2.6赋值语句和赋值表达式

1. 要点提示：赋值语句将一个值指定给一个变量。声明变量后，可以使用赋值语句给它赋一个值。

Variable=expression;（变量=表达式）

表达式表示涉及值，变量和操作符的一个运算，它们组合在一起计算出一个新值。

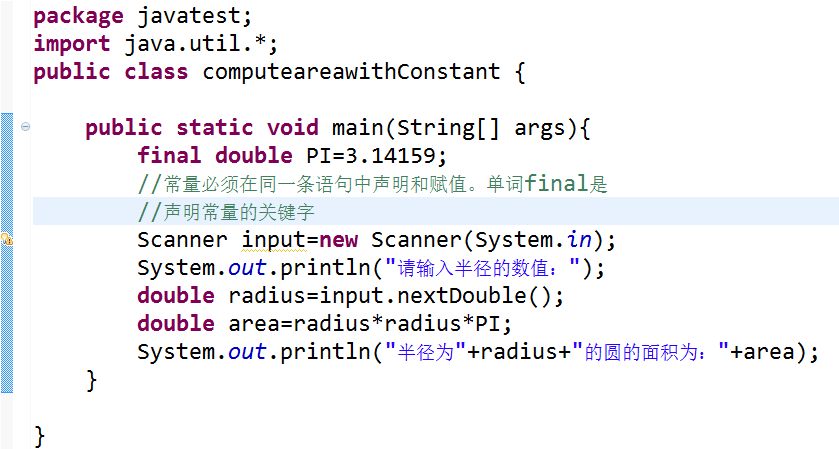
2..注意：在赋值语句中，左边变量的数据类型必须与右边值的数据类型兼容。例如，int x=1.0

是非法的，因为x的数据类型是整形int。

## 2.7命名常量

1. 要点提示：命名常量是一个代表不变值的标识符。一个变量的值在程序执行的过程中可能会发生变化，但是命名常量（常量），则表示从不改变的永久数据。

2. 使用常量的三个好处：（1）不必重复输入同一个值；（2）如果必须修改常量值，只需在源代码中的一个地方做改动；（3）给常量赋一个描述性的名字会提高程序的易读性。



## 2.8命名习惯

1. 要点提示：严格遵循Java的命名习惯可以让你的程序易于理解，以及避免错误。

2. 命名规范：

（1）使用小写字母命名变量和方法。如果一个名字包含多个单词，就将他们连在一起，第一个单词的字母小写，而后每个单词的首字母大写，例如，变量radius area 以及方法print。

（2）类名中的每个单词首字母大写，例如，类名Computer和System。

（3）大写常量中的所有字母，两个单词间用下划线连接，例如，常量PI 和常量MAX\_VALUE。

## 2.9数值数据类型和操作

1. 要点提示：Java针对整数和浮点数有六种数值类型，以及+ - \* / 和%等操作。

### 2.9.1数值类型

1. Java使用的四种类型的整数：byte short int 和long。

2. Java使用两种类型的浮点数：float和double。Double是float型的两倍。所以，double型又称“双精度”，而float称为“单精度”。通常情况下，应该使用double类型，因为它比float更精确。

### 2.9.2从键盘读取数值

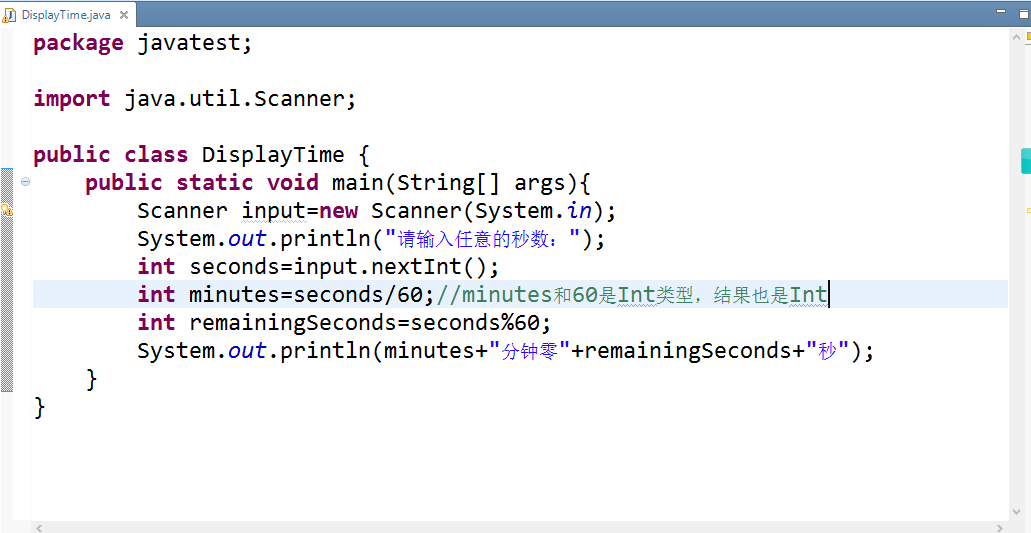
|  |  |
| --- | --- |
| 方法 | 描述 |
| nextByte( ) | 读取一个byte类型的整数 |
| nextShort( ) | 读取一个short类型的整数 |
| nextInt( ) | 读取一个int类型的整数 |
| nextLong( ) | 读取一个long类型的整数 |
| nextFloat( ) | 读取一个float类型的数 |
| nextDouble( ) | 读取一个double类型的数 |

Scanner对象的方法

### 2.9.3数值操作符

1. 当除法的操作数都是整数时，除法的结果就是整数，小数部分被舍去。例如：5/2的结果是2而不是2.5，而-5/2的结果是-2而不是-2.5。为了实现浮点数的除法，其中一个操作数必须是浮点数。例如：5.0/2=2.5。

2. 操作符%，求余或者取模操作符，可以求得除法的余数。通常用在正整数上，实际上它也可以用于负整数和浮点值。只有当被除数是负数时，余数才是负的。例如：-7%3结果是-1。



### 2.9.4幂运算

1. 使用方法Math.pow(a,b)来计算ab。pow方法定义在Java API中的Math类。

例如：System.out.println(Math.pow(2,3)); //结果为8

## 2.10数值型直接量

1. 要点提示：数值型直接量是一个程序中直接出现的常量值。

例如，下面语句中34和0.305都是直接量：

Int numberOfYears=34;

Double weight=0.305;

### 2.10.1整型直接量

1. 只要整型直接量与整形变量相匹配，就可以将整型直接量赋值给该整型变量。如果直接量太大，超出该变量的存储范围，就会出现编译错误。

例如：语句byte=128就会造成编译错误，因为byte型变量存不下128（注意：byte型变量的范围是-128-127）。

1. 默认情况下，整型直接量是一个十进制整数。要表示一个二进制的整数直接量，使用0b或者0B开头；表示一个八进制整型直接量，就用0开头，而要表示一个十六进制的整数直接量，就用0x或0X开头。

例如：

System.out.println(0B1111); //Display 15

System.out.println(07777); //4095

System.out.println(0XFFFF); //65535

### 2.10.2浮点型直接量

1. 浮点型直接量带小数点，默认情况下是double类型的。可以在数字后面加字母f或F表示该数为float类型直接量，也可以在数字后边加d或者D表示double型直接量。

例如：可以使用100.2f或100.2F表示float型值，用100.2d或100.2D表示double型值。

2. 注意double型值比float型值更精确。一个float值有7到8位小数，一个double值有15到17位小数。

### 2.10.3科学记数法

1. 浮点型直接量也可以用a\*10b形式的科学记数法表示。

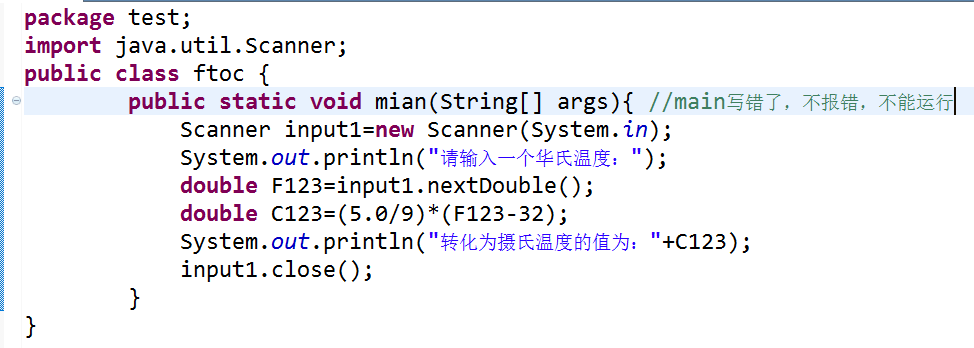
例如123.5=1.235\*102。

一种特定的语法可以用于表示科学记数法的数值。

例如1.235\*102可以写成1.235E2或者1.235E+2。其中，E或者e可以大写也可以小写。

## 2.11表达式求值以及操作符优先级

1. Java表达式的求值和数学表达式求值是一样。

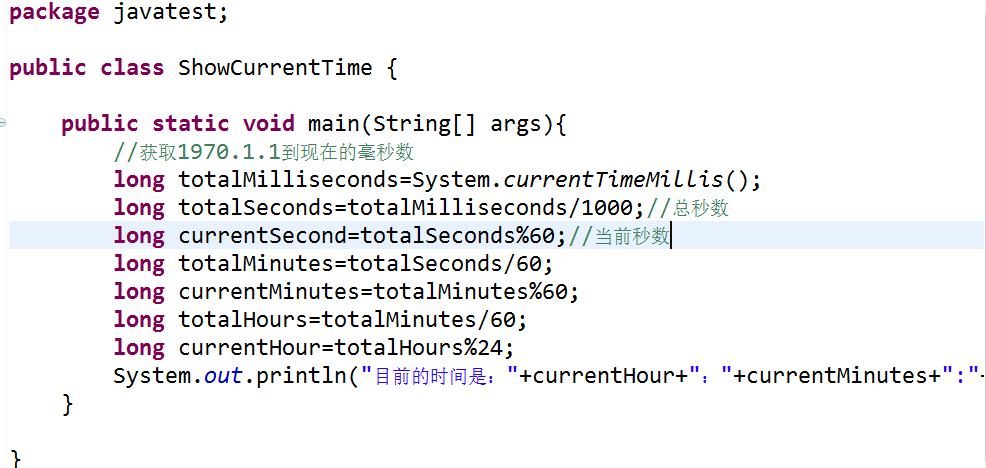


5.0/9而不是5/9，因为在Java中5/9的结果是0。

## 2.12实例学习：显示当前时间

1. 要点提示：可以通过调用System.currentTimeMillis( )返回当前时间。

2. System类中的方法currentTimeMillis返回从GMT 1970年1月1日00：00：00开始到当前时刻的毫秒数。



## 2.13增强赋值操作符

1.要点提示：操作符+-\*/可以结合赋值操作符形成增强操作符。

|  |  |  |  |
| --- | --- | --- | --- |
| 操作符 | 名称 | 示例 | 等价于 |
| += | 加法赋值操作符 | i+=8 | i=i+8 |
| -= | 减法赋值操作符 | i-=8 | i=i-8 |
| \*= | 乘法赋值操作符 | i\*=8 | i=i\*8 |
| /= | 除法赋值操作符 | i/=8 | i=i/8 |
| %= | 求余赋值操作符 | i%=8 | i=i%8 |

简捷赋值操作符

## 2.14自增和自减操作符

1. 要点提示：自增操作（++）和自减操作（--）是对变量进行加1或者减1的操作。

## 2.15数值类型转换

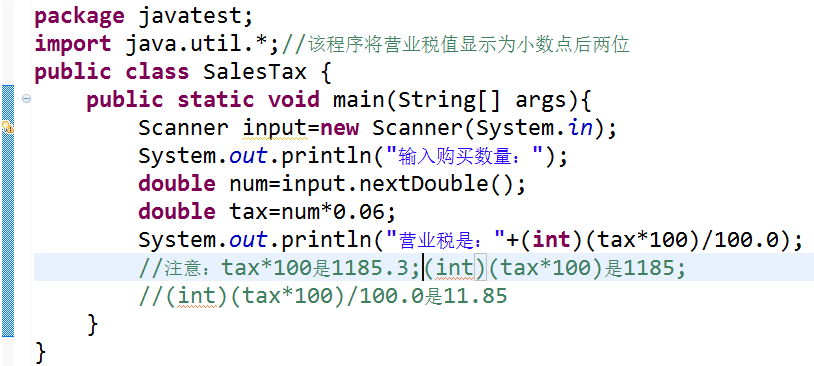
1. 要点提示：通过显式转换，浮点数可以被转换成整数。

2. 类型转换是将一种数据类型转换成另一种数据类型的操作。将一个小范围类型的变量转换为大范围类型的变量称为“拓宽类型”，将大范围类型的变量转换为小范围类型的变量称为“缩窄类型”。

例：System.out.println((int)1.7);

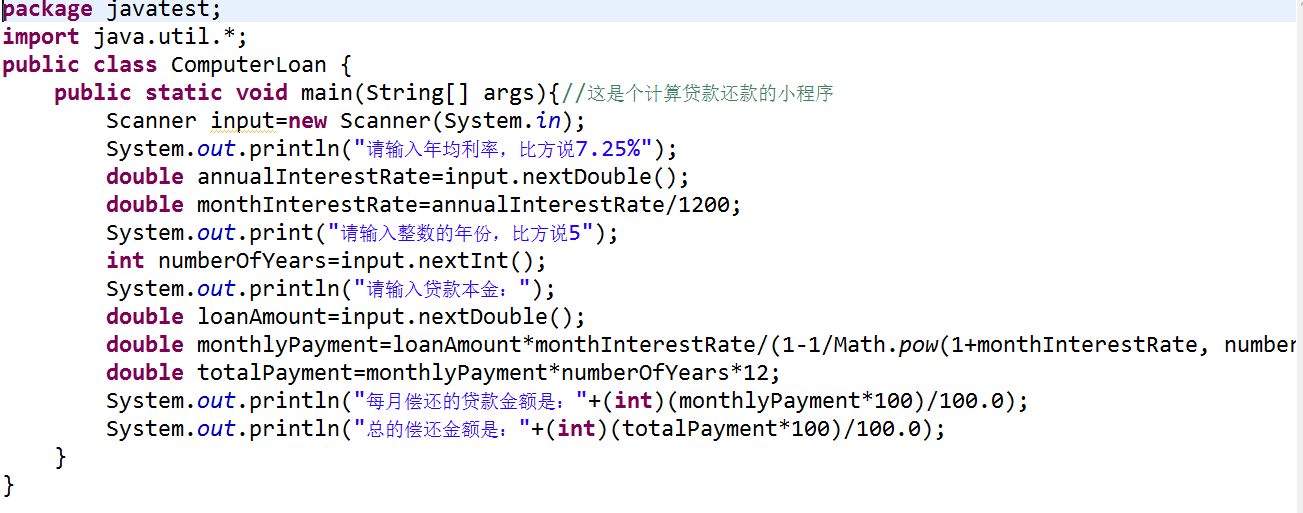
显示结果为1。当double类型被转换为int类型时，小部分被截去。

3.使用类型转换时必须小心，丢失的信息也许会导致不精确的结果。

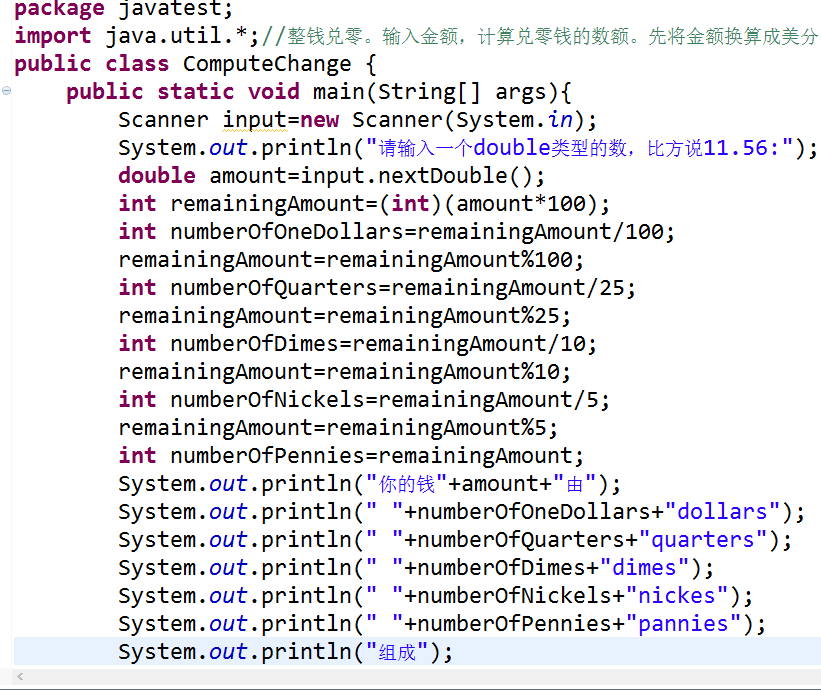


## 2.16软件开发过程

需求规范—分析—设计—实现—测试—部署—维护



## 2.17零钱找兑



## 2.18常见的错误和陷阱

1.要点提示：常见的基础编程错误经常涉及未声明变量，未初始化变量，整数溢出，超出预期的整数除法，以及整数取整错误。

**常见错误一**：未声明，未初始化的变量和未使用的变量。

变量必须在使用之前声明为一个类型并赋值。

**常见错误二**：整数溢出。

数字以有限的位数存储。当一个变量被赋予一个过大（以存储大小而言）的值，以致无法存储该值，这称为“溢出”。

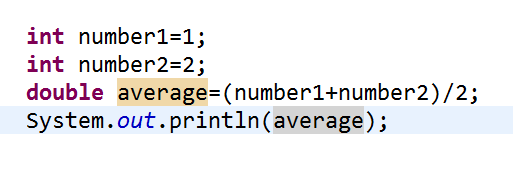
**常见错误三：**取整错误

一个取整错误，也成为“凑整错误”，是在计算得到的数字的近似值和确切的算数值之间的不同。例如，如果保留三位小数位数，1/3近似等于0.333，如果保留7位，近似值0.3333333。因为一个变量保存的位数是有限的，因此取整错误是无法避免的。

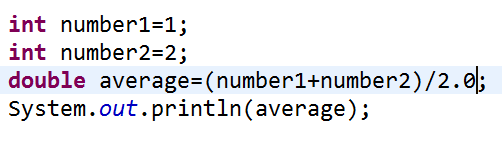
**常见错误四**：超出预期的整数除法

Java使用同样的除法操作符来执行整数和浮点数的除法。当两个操作数是整数时，/操作符执行一个整数除法，操作的结果是整数，小数部分被截去。要强制两个整数执行一个浮点数除法时，将其中一个整数转换为浮点数值。

例如：下面a中的代码显示平均值为1，b中的代码显示平均值为1.5。

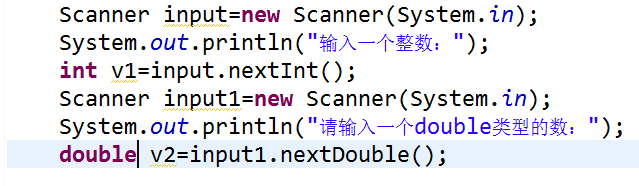


(a)

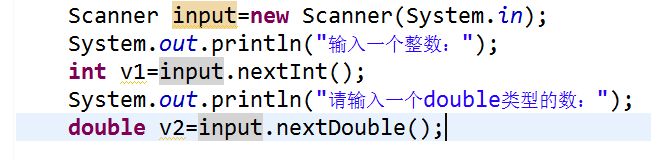


(b)

**常见陷阱： 冗余输入错误**



（a）



(b)

（a）代码没有错，但是效率低。它创建了两个不必要的输入对象，可能会导致一些不易发现的错误。代码可以改为（b）样式。