Java程序设计

Java Programming Design

毛斐巧

感谢：教材《Java大学实用教程》的作者和其他老师提供PowerPoint讲义等资料！
说明：本课程所使用的所有讲义，都是在以上资料上修改的。

易出现的问题

- ！！：一个类文件中定义多个类
- ＸＸ：Test.java中定义了类Test、Menu和Customer
- 通常一个类文件（.java)中只定义一个类，除了内部类或线程类特殊情况

运算符表达式及控制语句

主要内容

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

3.1 算术运算符和算术表达式

- (1) 加、减运算符
  - 符号：+、-，双目运算符
  - 结合性：从左到右
  - 操作元：整型或浮点型数据
  - 优先级：4级

- (2) 乘、除和求余运算符
  - 符号：*、/、%，双目运算符
  - 结合性：从左到右
  - 操作元：整型或浮点型数据
  - 优先级：3级

Numeric Operators(数值运算符)

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 – 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

%(只适用于整数)

5 % 2 yields 1 (the remainder of the division)

3.1 算术运算符和算术表达式

- (3) 自增、自减运算符
  - 符号：++、--，单目运算符
  - 优先级：2级

  - 运算符可以放在操作元之前，也可以放在操作元之后，但操作元必须是一个整型或浮点型变量（不能是常量或表达式）
  - 运算符的作用：使变量的值增1或减1
    - ++x, --x：在使用x之前，先使x的值加（减）1
    - x++, x--：在使用x之后，才使x的值加（减）1

    - 例子：如果 x的原值是5，则
      - 对于y=++x; y的值为6
      - 对于y=x++; y的值为5，然后x的值才变为6

自增自减运算符

```
int i = 10;
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i;
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;
int newNum = 10 * i;
```

3.1 算术运算符和算术表达式

- (4) 算术表达式
  - 用算术符号和操作元连接起来的符合Java语法规则的式子
  - 例子：x+2*y-30+3*(y+5) -12+n+(--n)

3.1 算术运算符和算术表达式

- (5) 算术混合运算的精度
  - 精度从“低”到“高”排列的顺序是
    - byte, short, int, long, float, double
  - Java将按运算符两边的操作元的最高精度保留结果的精度
  - 例子：
    - 5/2的结果是2
    - **5.0/2或5.0f/2的结果是2.5**
  - char型数据与整型数据运算的结果的精度是int
    - 例子：char ch=(char)(‘H’+k);

Outline

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

3.2 关系运算符和关系表达式

- 关系运算符用来比较两个值的关系
- 关系运算符的运算结果是boolean型数据（true, false）

3.2 关系运算符和关系表达式

- (1) 大小关系运算符
  - 符号：>、>=、<、<=，双目运算符
  - 操作元：数值型的常量、变量或表达式
  - 优先级：6级
  - 例子：10>20-17相当于10>(20-17)
- (2) 等、不等关系
  - 符号：==、!=，双目运算符
  - 优先级：7级
  - 注：不要将赋值运算符“=”与等号运算符“==”混淆

3.2 关系运算符和关系表达式

- (3) 关系表达式
  - 运算结果为一个布尔型数值，通过关系运算符连接的式子
  - 例子：24>18，(x+y+z)>30+x

Outline

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

3.3 逻辑运算符和逻辑表达式

- 用来实现boolean型数据的逻辑“与”、“或”和“非”运算
- 用逻辑运算符连接的式子是逻辑表达式
- 运算结果是boolean型数据

3.3 逻辑运算符和逻辑表达式

- (1) 逻辑“与”和逻辑“或”
  - 符号：&&、||，双目运算符
  - 操作元：boolean型的变量或求值结果是boolean型数据的表达式
  - 结合性：从左到右
  - 优先级：&&和||的级别分别是11和12级
  - 运算法则：
    - &&：当2个操作元的值都是true时，运算结果是true，否则是false
    - ||：当2个操作元的值都是false时，运算结果是false，否则是true
- (2) 逻辑“非”
  - 符号：!，单目运算符
  - 运算级别：2级
  - 结合性：从右到左
  - 例子：!!X 相当于!(!X)

3.3 逻辑运算符和逻辑表达式

- (3) 逻辑表达式
  - 结果为boolean型的变量或表达式可以通过逻辑运算符形成逻辑表达式
  - 例子：24>18&&4<0，x!=0||y!=0

Outline

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

3.4 赋值运算符和赋值表达式

- 符号：=，双目运算符
- 左面的操作元必须是变量，不能是常量或表达式
- 结合性：从右到左
- 优先级：14级

3.4 赋值运算符和赋值表达式

- 符号：=，双目运算符
- 左面的操作元必须是变量，不能是常量或表达式
- 结合性：从右到左
- 优先级：14级

Augmented Assignment Operators（扩充赋值算子）

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`

Outline

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

3.5 移位运算符

- 移位运算符用来对二进制位进行操作，分为左移位操作和右移位操作

3.5 移位运算符

- (1) 左移位运算符
  - 符号：<<，双目运算符
  - 左面的操作元：被移位数
  - 右面的操作元：移位量，操作元必须是整型类型的数据

  - 效果（a<<n）
    - 将a的所有位都左移n位，每左移一个位，左边的高阶位上的0或1被移出丢弃，并用0填充右边的低位
    - 不断左移位的结果是0

3.5 移位运算符

- 对于byte或short型数据，a<<n的运算结果是int型精度
  - 当进行a<<2运算时，系统首先将a升级为int型数据，对于正数，在高位用0填充；负数用1填充，然后再进行移位运算
  - 例子：byte a=-8; byte b=(byte)(a<<1);
    - 1111 1000
    - 1111 1111 1111 1111 1111 1111 1111 1000
    - 1111 1111 1111 1111 1111 1111 1111 0000
    - 1111 0000
- 在进行a<<n运算时
  - 如果a是byte、short或int型数据，系统总是先计算出**n%32**的结果m，然后再进行a<<m运算
  - 如果a是long型数据，系统总是先计算出**n%64**的结果m，然后再进行a<<m运算

3.5 移位运算符

- (2) 右移位运算符
  - 符号：>>，双目运算符
  - 左面的操作元：被移位数
  - 右面的操作元：移位量，操作元必须是整型类型的数据

  - 效果（a>>n）：
    - 将a的所有位都右移n位，每右移一个位，右边的低阶位被移出丢弃，并用0或1填充左边的高位，a是正数时用0填充，负数时用1填充
    - 正数不断右移位的结果是0，负数不断右移位的结果是-1

3.5 移位运算符

- 对于byte或short型数据，a>>n的运算结果是int型精度

- 在进行a>>n运算时
  - 如果a是byte、short或int型数据，系统总是先计算出**n%32**的结果m，然后再进行a>>m运算
  - 如果a是long型数据，系统总是先计算出**n%64**的结果m，然后再进行a>>m运算

3.5 移位运算符

- 【例子】

```
import java.util.*;
public class Example3_1
{
    public static void main (String args[])
    {
        Scanner reader=new Scanner(System.in);
        System.out.println("输入待移位的int型整数:");
        int x = reader.nextInt();
        System.out.println("输入移位量:");
        int n = reader.nextInt();
        System.out.println("左移位的结果:"+(x<<n));
        System.out.println("右移位的结果:"+(x>>n));
    }
}
```

```
输入待移位的int型整数:
-8
输入移位量:
1
左移位的结果:-16
右移位的结果:-4
```

Outline

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

3.6 位运算符（bitwise operators）

- (1) “按位与”运算符
  - 符号：&，双目运算符
  - 效果：对两个整型数据a, b按位进行运算，运算结果是一个整型数据c
  - 运算法则：
    - 如果a, b两个数据对应位都是1，则c的该位是1，否则是0
    - 如果b的精度高于a，那么结果c的精度和b相同

3.6 位运算符（bitwise operators）

- (2) “按位或”运算符
  - 符号：|，双目运算符
  - 效果：对两个整型数据a, b按位进行运算，运算结果是一个整型数据c
  - 运算法则：
    - 如果a, b两个数据对应位都是0，则c的该位是0，否则是1
    - 如果b的精度高于a，那么结果c的精度和b相同

3.6 位运算符（bitwise operators）

- (3)“按位非”运算符
  - 符号：~，单目运算符
  - 效果：对一个整型数据a按位进行运算，运算结果是一个整型数据c
  - 运算法则：如果a对应位是0，则c的该位是1，否则是0

3.6 位运算符（bitwise operators）

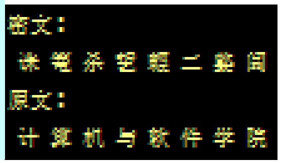
- (4) “按位异或”运算符
  - 符号：^，双目运算符
  - 效果：对两个整型数据a, b按位进行运算，运算结果是一个整型数据c
  - 运算法则：
    - 如果a, b两个数据对应位相同，则c的该位是0，否则是1
    - 如果b的精度高于a，那么结果c的精度和b相同
  - a^0=a, a^1=~a
  - a^a=0
  - a^b^b=a

3.6 位运算符（bitwise operators）

- 【例子】

```
public class Example3_2
{
       public static void main(String args[])
       {
              char a[]={'计','算','机','与','软','件', '学', '院'};
              char secret='z';
              for(int i=0;i<a.length;i++)
              {
                   a[i]=(char)(a[i]^secret);
              }
              System.out.printf("密文:\n");
              for(int i=0;i<a.length;i++)
              {
                   System.out.printf("%3c",a[i]);
              }

              for(int i=0;i<a.length;i++)
              {
                   a[i]=(char)(a[i]^secret);
              }
              System.out.printf("\n原文:\n");
              for(int i=0;i<a.length;i++)
              {
                   System.out.printf("%3c",a[i]);
              }
       }
}
```



利用“异或”运算，对字符进行加密并输出密文，然后再解密

3.6 位运算符（bitwise operators）

- 位运算符也可以操作逻辑型数据
  - 当a, b都是true时，a&b是true，否则a&b是false
  - 当a, b都是false时，a|b是false，否则a|b是true
  - 当a是true时，~a是false；当a是false时，~a是true

Outline

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

3.7 条件运算符

- 符号：?:，3目运算符
- 用法：op1?op2:op3，op1的值必须是boolean型数据

- 运算法则：
  - 当op1的值是true时，op1?op2:op3的运算结果是op2的值
  - 当op1的值是false时，op1?op2:op3的运算结果是op3的值

```
if (x > 0)
  y = 1
else
  y = -1;

等价于
y = (x > 0) ? 1 : -1;
```

```
if (num % 2 == 0)
  System.out.println(num + “is even”);
else
  System.out.println(num + “is odd”);

等价于
System.out.println(
  (num % 2 == 0)? num + “is even” :
  num + “is odd”);
```

Outline

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

3.8 instanceof运算符

- 双目运算符　运算结果是一个布尔型的值
- 左面的操作元是一个对象，右面是一个类
- 当左面的对象是右面的类创建的对象时，该运算的结果是true，否则是false

- 例：

```
boolean f = rectangleOne instanceof Rect;
```

Operator Precedence（运算符的优先级）

- var++, var--
- +, - (一元加、减), ++var,--var
- (type) 类型转换
- ! (非)
- *, /, % (乘, 除, 取余)
- +, - (二元加、减)
- <, <=, >, >= (关系运算符)
- ==, !=; (等、不等运算符)
- ^ (异或)
- && (条件与)
- || (条件或)
- =, +=, -=, *=, /=, %= (赋值运算符)

Outline

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

3.9 一般表达式

- Java的一般表达式：用运算符及操作元连接起来的符合Java规则的式子，简称表达式

- 一个Java表达式必须能求值，即按着运算符的计算法则，可以计算出表达式的值

- 例子

```
public class LearningJava
{
    public static void main(String[] args)
    {
        int x=1,y=-2,n=10;
        int z = x+y+(--n)*(x>y&&x>0?(x+1):y);
        System.out.println(z);
    }
}
```

```
17
```

Outline

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

3.10 语句概述

- Java里的语句可分为以下5类
  - (1) 方法调用语句，例如reader.nextInt();
  - (2) 表达式语句，例如x=23;
  - (3) 复合语句，可以用“{”和“}”把一些语句括起来构成复合语句，一个复合语句也称作一个代码块(block)
  - (4) 控制语句，包括条件分支语句、循环语句和跳转语句
  - (5) package语句和import语句

Outline

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

3.11 分支语句

- 1.条件分支语句
- (1) if-else语句
  - if-else语句是Java中的一条语句，由一个“if”、“else”和两个复合语句按一定格式构成，if-else 语句的格式如下

```
if(表达式)
{若干语句}
else
{若干语句}
```

  - if后面()内的表达式的值必须是boolean型的。如果表达式的值为true，则执行紧跟着的复合语句；如果表达式的值为false，则执行else后面的复合语句。

3.11 分支语句

- (2) 多条件if-else if-else语句
  - 程序有时需要根据多个条件来选择某一操作，这时就可以使用if-else if-else语句。

3.11 分支语句

- 【例子】

```
import java.util.*;
public class Example3_3
{
    public static void main (String args[])
    {
        Scanner reader=new Scanner(System.in);
        double a=0,b=0,c=0;
        System.out.print("输入边a:"); a=reader.nextDouble();
        System.out.print("输入边b:"); b=reader.nextDouble();
        System.out.print("输入边c:"); c=reader.nextDouble();
        if(a+b>c && a+c>b && b+c>a)
        {
            if(a*a==b*b+c*c||b*b==a*a+c*c||c*c==a*a+b*b)
            {
               System.out.printf("%-8.3f%-8.3f%-8.3f构成是直角三角形",a,b,c);
            }
            else if(a*a<b*b+c*c&&b*b<a*a+c*c&&c*c<a*a+b*b)
            {
               System.out.printf("%-8.3f%-8.3f%-8.3f构成锐角三角形",a,b,c);
            }
            else
            {
               System.out.printf("%-8.3f%-8.3f%-8.3f构成钝角三角形",a,b,c);
            }
        }
        else
        {
            System.out.printf("%f,%f,%f不能构成三角形",a,b,c);
        }
    }
}
```

用户在键盘输入3个数，程序判断这3个数能构成什么形状的三角形

注意

```
if i > 0 {
  System.out.println("i is positive");
}
```

(a) Wrong

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(b) Correct

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(a)

Equivalent

```
if (i > 0)
  System.out.println("i is positive");
```

(b)

多重选择if 语句例子

```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

(a)

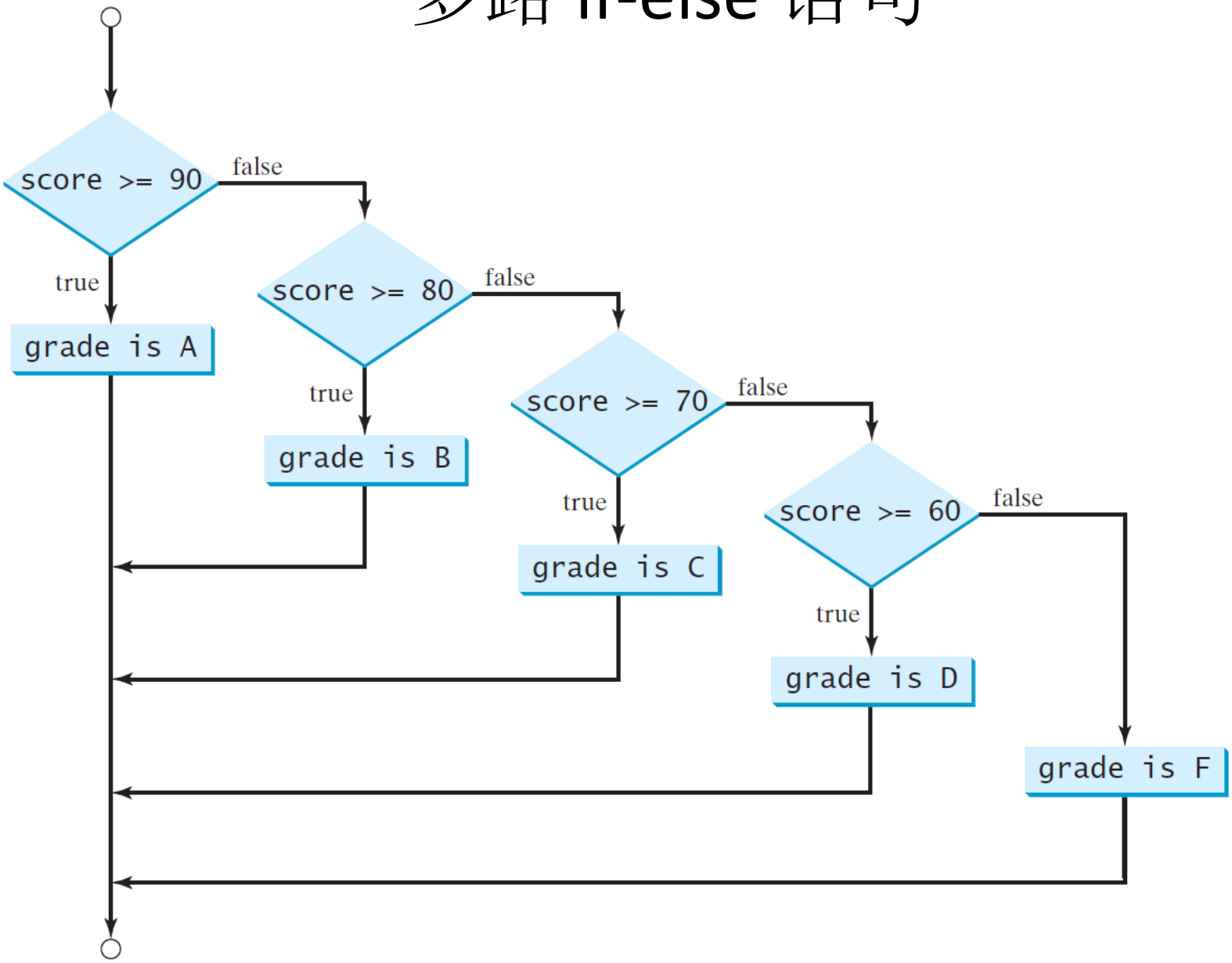
Equivalent

This is better

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

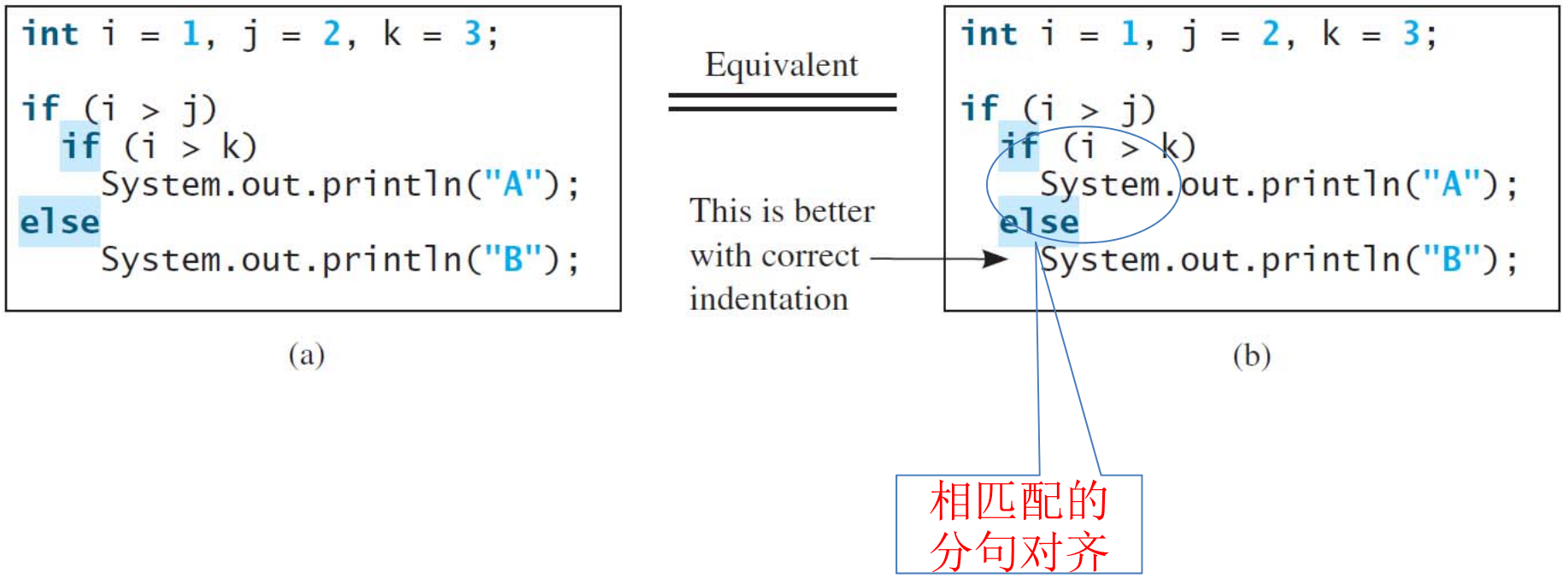
(b)

多路 if-else 语句


score >= 90
false
true
grade is A
score >= 80
false
true
grade is B
score >= 70
false
true
grade is C
score >= 60
false
true
grade is D
grade is F

注意

The else clause（分句） matches the most recent if clause in the same block.



注意

Nothing is printed from the preceding statement. To force the else clause to match the first if clause, you must add a pair of braces: （使用花括号达到匹配目的）

```
int i = 1;
int j = 2;
int k = 3;
if (i > j) {
  if (i > k)
    System.out.println("A");
}
else
  System.out.println("B");
```

This statement prints B.

常见错误

Adding a semicolon（;） at the end of an if clause is a common mistake.

```
if (radius >= 0);
{
  area = radius*radius*PI;
  System.out.println(
    "The area for the circle of radius " +
    radius + " is " + area);
}
```

Wrong

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error. （这类错误不易发现）

This error often occurs when you use the next-line block style.

Tips(小提示)

```
if (number % 2 == 0)
  even = true;
else
  even = false;
```

(a)

Equivalent

```
boolean even
  = number % 2 == 0;
```

(b)

警示

```
if (even == true)
  System.out.println(
    "It is even.");
```

(a)

Equivalent

```
if (even)
  System.out.println(
    "It is even.");
```

(b)

3.11 分支语句

- 2.switch开关语句
  - switch语句是多分支的开关语句，它的一般格式定义如下：

```
switch(表达式)
{
        case 常量值1:
                若干个语句
                 break;
        case 常量值2:
                若干个语句
                break;
                ...
        case 常量值n:
                若干个语句
                break;
        default:
                若干语句
}
```

3.11 分支语句

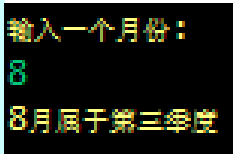
- switch语句中表达式的值必须是整型或字符型；常量值1到常量值n也必须是整型或字符型
- switch语句首先计算表达式的值，如果表达式的值和某个case 后面的常量值相同，就执行该case里的若干个语句，直到碰到break语句为止。若没有任何常量值与表达式的值相同，则执行default后面的若干个语句。其中，default是可有可无的，如果它不存在，并且所有的常量值都和表达式的值不相同，那么switch语句就不会进行任何处理

- 注意：在同一个switch语句中，case后的常量值必须互不相同

3.11 分支语句

- 【例子】

```
import java.util.*;
public class Example3_4
{
     public static void main (String args[])
     {
          Scanner reader=new Scanner(System.in);
          System.out.println("输入一个月份:");
          int n=reader.nextInt();
          switch(n)
          {
               case 1:
               case 2:
               case 3: System.out.printf("%d月属于第一季度",n); break;
               case 4:
               case 5:
               case 6: System.out.printf("%d月属于第二季度",n); break;
               case 7:
               case 8:
               case 9: System.out.printf("%d月属于第三季度",n); break;
               case 10:
               case 11:
               case 12: System.out.printf("%d月属于第四季度",n); break;
               default: System.out.printf("%d不代表月份",n);
          }
      }
}
```



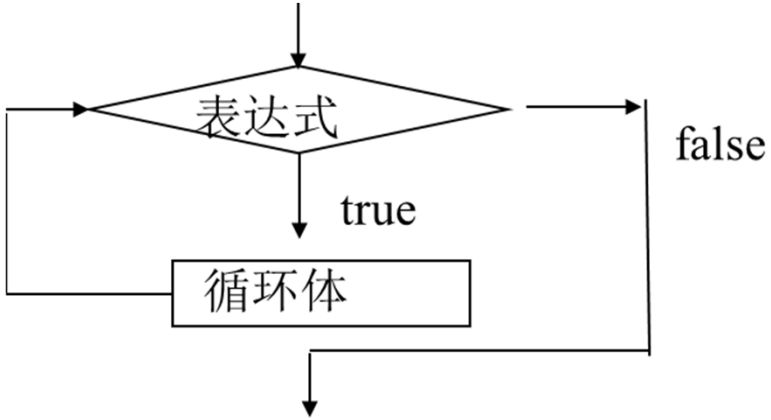
用户在键盘输入一个代表月份的整数，程序输出是该月是在年度的第几季度

Outline

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

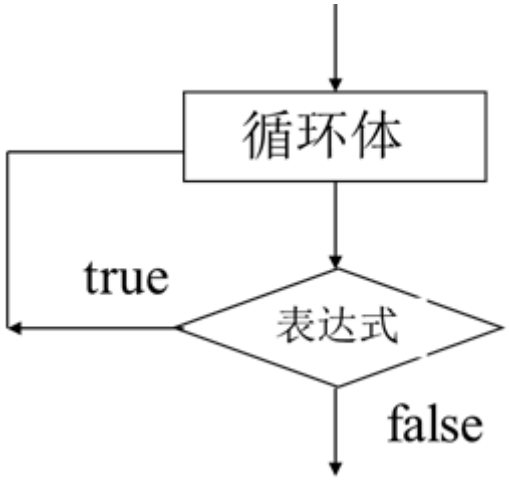
3.12 循环语句

- 1.while循环
  - while语句由关键字while、括号中的一个求值为boolean型数据的表达式和一个复合语句组成，其中的复合语句称作循环体
  - 循环体只有一条语句时，大括号“{}”可以省略，但最好不要省略，以便增加程序的可读性。
  - 表达式称作循环条件
  - while语句的执行规则



3.12 循环语句

- 2.do-while循环
  - do-while循环和while循环的区别是，do-while的循环体至少被执行一次



3.12 循环语句

- 【例子】

```
public class Example3_5
{
    public static void main (String args[])
    {
        double sum=0,item=1;
        int i=1;
        while(i<=1000)
        {
            sum=sum+item;
            i++;
            item=item*(1.0/i);
        }
        sum=sum+1;
        System.out.println("e="+sum);

        sum=0;
        i=1;
        item=1;
        do{
            sum=sum+item;
            i++;
            item=item*(1.0/i) ;
        }while(i<=1000);
        sum=sum+1;
        System.out.println("e="+sum);
    }
}
```

```
e=2.7182818284590455
e=2.7182818284590455
```

分别用while和do-while循环计算常数e的近似值：e=1+1+1/2!+1/3!+…

3.12 循环语句

- 3.for循环
  - for语句的一般格式

```
for (表达式1; 表达式2; 表达式3)
{若干语句}
```

  - for语句由关键字for，括号中用分号分割的3个表达式，以及一个复合语句组成。

  - 表达式1：负责完成变量的初始化
  - 表达式2：值为boolean型数据的表达式，称为循环条件
  - 表达式3：修改变量，改变循环条件

3.12 循环语句

- 【例子】

```
public class Example3_6
{
      public static void main(String args[])
      {
            int sum,i,j;
            for(i=1;i<=1000;i++)
            {
                  for(j=1,sum=0;j<=i/2;j++)
                  {
                        if(i%j==0)
                        {
                              sum=sum+j;
                        }
                  }
                  if(sum==i)
                  {
                        System.out.printf("%8d是一个完数%n",i);
                  }
            }
      }
}
```

```
6是一个完数
28是一个完数
496是一个完数
```

Outline

- 3.1 算术运算符和算术表达式
- 3.2 关系运算符和关系表达式
- 3.3 逻辑运算符和逻辑表达式
- 3.4 赋值运算符和赋值表达式
- 3.5 移位运算符
- 3.6 位运算符
- 3.7 条件运算符
- 3.8 instanceof运算符
- 3.9 一般表达式
- 3.10 语句概述
- 3.11 分支语句
- 3.12 循环语句
- 3.13 跳转语句

3.13 跳转语句（branching）

- 跳转语句是指用关键字break或continue加上分号构成的语句
  - **break语句**：如果在某次循环体的执行中执行了break语句，那么整个循环语句就结束
  - **continue语句**：如果在某次循环体的执行中执行了continue语句，那么本次循环就结束，即不再执行本次循环中continue语句后面的语句，而转入下一次循环

3.13 跳转语句（branching）

- 【例子】

```
public class Example3_7
{
      public static void main(String args[])
      {
           int sum=0,i=0,max=8888,number=7;
           while(true)
           {
                i++;
                sum=sum+i;
                if(sum>=max)
                    break;
           }
           System.out.println("1+2+...n<"+max+"的最大整数n是:"+(i-1));

           for(i=1,max=200,sum=0;i<=max;i++)
           {
                if(i%number!=0)
                {
                    continue;
                }
                sum=sum+i;
           }
           System.out.println(max+"内能被"+number+"整除的数字之和:"+sum);
    }
}
```

```
1+2+...n<8888的最大整数n是:132
200内能被7整除的数字之和:2842
```

3.13 跳转语句（branching）

- 【例子】

```
import java.util.Scanner;
public class Example3_8
{
	public static void main(String args[])
	{
		int n,start,end,middle;
		int a[]={-2,1,4,5,8,12,17,23,45,56,90,100};
		start=0; end=a.length;
		middle=(start+end)/2;
		int count=0;
		Scanner reader=new Scanner(System.in);
		System.out.print("请输入一个整数:");
		n=reader.nextInt();
		while(n!=a[middle])
		{
			if(n>a[middle]){start=middle;}
			else if(n<a[middle]){end=middle;}
			middle=(start+end)/2;
			count++;
			if(count>a.length/2)
				break;
		}
		if(count>a.length/2)
			System.out.println(n+"不在数组中");
		else
			System.out.println(n+"是数组中的第"+middle+"个元素");
	}
}
```

```
请输入一个整数:100
100是数组中的第11个元素
```

折半法查找一个整数是否在一个排序的int类型数组中

小节

- 算术运算符、关系运算符、逻辑运算符
- 赋值运算符
- 移位运算符
- 位运算符
- 条件运算符
- instanceof运算符
- 一般表达式、语句概述、分支语句、循环语句、跳转语句

- 没必要记忆运算符的优先级别，可以在编程序时尽量使用括号“()”来实现想要的运算次序，以免产生难以阅读或含糊不清的计算顺序。
- “()”也是一种运算符，它的级别最高。

小节

Operator Precedence

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html

小节

- http://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html
- 有时间请看一下上面链接中的“Lesson: Language Basics”内容