操作符

1. 代码观察

观察以下代码

```
1  int a = 6--;
2  System.out.println(a);
```

问题：上面代码是否有错误，如果没有，程序结果是多少?

答案：

在Java中，这行代码也会导致编译错误。在Java中，递减运算符`--`也是一个单目运算符，用于将变量的值减1。然而，与C语言不同的是，Java不允许将数字常量作为左值来进行赋值操作。因此，这行代码会被解释为试图将数字6赋值给一个常量，从而导致编译错误。

如果您想要递减一个变量并将结果赋给另一个变量，可以使用以下代码：

```
int a = 6;
int b = --a;
```

这将首先将6赋值给变量a，然后将a的值减1，并将结果赋值给变量b。现在，变量a的值为5，变量b的值为5。

2. 代码分析

分析以下代码的运行结果【重点复习原码 反码 补码】

```
System.out.println(5&9);
System.out.println(5|9);
System.out.println(5^9);
System.out.println(~-5);
System.out.println(5<<2);
System.out.println(-5<<2);
System.out.println(-5>>2);
System.out.println(-5>>>2);
System.out.println(5>>2);
System.out.println(5>>34);
System.out.println(97=='a');
System.out.println(5.0==5);
System.out.println(4>5 ^ 'c'>'a');
System.out.println((int)(char)(byte)-1);
```

答案：

这段代码的运行结果如下：

```
/* ************************
          * 5: 0000 0101
          * &
          * 9: 0000 1001
          * -------------
          *   0000 0001
          * ************************
          */
System.out.println(5&9); // 1

/* ************************
          *  5: 0000 0101
          * |
          *  9: 0000 1001
          * -------------
          *     0000 1101
          * ************************
          */
System.out.println(5|9); // 13

/* ************************
          *  5: 0000 0101
```

```
23              * ^
24              *  9: 0000 1001
25              * -------------
26              *     0000 1100
27              * ************************
28              */
29  System.out.println(5^9); //12
30
31  /* ************************
32              *  -5: 1000 0101
33              * 反码: 1111 1010
34              * 补码: 1111 1011
35              * 取反 0000 0100
36              * ************************
37              */
38  System.out.println(~-5); // 4
39
40  /* ************************
41              *      5: 0000 0101
42              * 左移2位: 0001 0100
43              * ************************
44              */
45  System.out.println(5<<2); // 20
46
47  /* ************************
48              * -5源码: 1000 0101
49              *   反码: 1111 1010
```

```
50              *   补码: 1111 1011
51              * 左移2位:1110 1100
52              *   反码:1110 1011
53              *   原码:1001 0100
54              * ************************
55              */
56  System.out.println(-5<<2); -20
57
58  /* ************************
59              * -5: 1000 0101
60              * 反码: 1111 1010
61              * 补码: 1111 1011
62              * 右移: 1111 1110
63              * 反码: 1111 1101
64              * 原码: 1000 0010
65              * ************************
66              */
67  System.out.println(-5>>2); // -2
68
69  /* ************************
70              *  -5: 1000 0000 0000 0000 0000
    0000 0000 0101
71              * 反码: 1111 1111 1111 1111 1111
    1111 1111 1010
72              * 补码: 1111 1111 1111 1111 1111
    1111 1111 1011
```

```
73            * 右移: 0011 1111 1111 1111 1111
    1111 1111 1111
74            * **************************
75            */
76  System.out.println(-5>>>2); // 1073741822
77
78  /**
79            * 5:  0000 0101
80            * 右移 0000 0001
81            */
82  System.out.println(5>>2); // 1
83
84  /**
85            * 5:  0000 0101
86            * 右移 0000 0001
87            *
88            */
89  System.out.println(5>>34); // 1
90
91  System.out.println(97 == 'a'); // true
92  System.out.println(5.0 == 5); // true
93
94  /**
95            * false^true
96            */
97  System.out.println(4 > 5 ^ 'c' > 'a'); //
    true
```

```
98
99  /**
100             *
101             * -1(int):     1000 0000 0000
    0000 0000 0000 0000 0001
102             *  反码:       1111 1111 1111
    1111 1111 1111 1111 1110
103             *  补码:       1111 1111 1111
    1111 1111 1111 1111 1111
104             *  (byte) -1: 1111 1111
105             *  反码:       1111 1110
106             *  原码:       1000 0001        -
    >-1
107             * -1(int):         1000 0000 0000
    0000 0000 0000 0000 0001
108             *  反码:       1111 1111 1111
    1111 1111 1111 1111 1110
109             *  补码:       1111 1111 1111
    1111 1111 1111 1111 1111
110             *  (char)-1 : 1111 1111 1111 1111
       ->65535
111             *  65535:     0000 0000 0000 0000
    1111 1111 1111 1111
112             *
113             */
114 System.out.println((int)(char)(byte)-1); //
    -1
```

下面是每个表达式的分析：

1. `System.out.println(5&9);`，按位与运算，结果为1（二进制0001）。
2. `System.out.println(5|9);`，按位或运算，结果为13（二进制1101）。
3. `System.out.println(5^9);`，按位异或运算，结果为12（二进制1100）。
4. `System.out.println(~-5);`，按位取反运算，结果为4。首先，-5的补码为11111011，取反后得到00000100，即4的补码。
5. `System.out.println(5<<2);`，左移运算，结果为20。将5的二进制数左移2位得到10100，即20。
6. `System.out.println(-5<<2);`，左移运算，结果为-20。由于Java中的移位运算符是对补码进行操作，因此-5的补码为11111011，将其左移两位得到11101100，即-20的补码。
7. `System.out.println(-5>>2);`，算术右移运算，结果为-2。由于Java中的算术右移运算符也是对补码进行操作，因此-5的补码为11111011，将其右移两位得到11111110，即-2的补码。
8. `System.out.println(-5>>>2);`，逻辑右移运

算，结果为1073741822。由于Java中的逻辑右移运算符是对无符号数进行操作，因此需要将-5的补码转换为无符号数的值。首先，-5的补码为11111011，在补码的基础上求其补码得到00000101，即5的补码，将其右移两位得到00000001，即1的补码，将其转换为十进制得到1，然后将其转换为有符号整数，得到1073741822。

9. `System.out.println(5>>2);`，算术右移运算，结果为1。将5的二进制数右移两位得到0001，即1。
10. `System.out.println(5>>34);`，算术右移运算，结果为1。右移34位相当于右移2位，因此结果与表达式9相同。
11. `System.out.println(97=='a');`，比较运算，结果为true。因为97是字符'a'的ASCII码。
12. `System.out.println(5.0==5);`，比较运算，结果为true。因为Java会将5转换为5.0，然后比较两个浮点数的值是否相等。
13. `System.out.println(4>5 ^ 'c'>'a');`，逻辑异或运算，结果为true。首先，'c'的ASCII码为99，'a'的ASCII码为97，因此'c'>'a'为true。然后，4>5为false。最后，false异或true的结果为true。
14. `System.out.println((int)(char)`

> `(byte)-1);`从较窄的整型转换成较宽的整型时的符号扩展行为：如果最初的数值类型是有符号的，那么就执行符号扩展；如果它是char，那么不管它将要被转换成什么类型，都执行零扩展。byte类型的-1的补码为11111111，拓展为char类型，补码为1111111111111111，拓展为整型，前面补16个0，得到结果为65535

3. 指定位置设置为1

问题描述 ：

将参数v的第n位置设为1，然后返回该结果值。

答案：

```
1  public class Test3_SetOne {
2    public static void main(String[] args) {
3        // 0b 0000 1010
4        int r = setBit(10,3);
5        System.out.println("r: " + r);
6    }
7
8    /*
```

```
9       将v的第n位置为1
10
11      假设:
12      v=10  0b 0000 1010
13        n=3      |0000 0100    <--    1左移可得到
14                 0000 1110 即可
15     */
16    public static int setBit(int v,int n) {
17        int mask = 1 << (n-1);
18        int t = mask | v;
19        return t;
20    }
21  }
```

首先，左移操作 `1 << (n - 1)` 可以将数字 `1` 左移 `n - 1` 位，得到一个只有第 n 位为 1 的二进制数，名为掩码（mask）。例如当 n=3 时，掩码为 4 (二进制 100)。然后，按位或运算符 `|` 可以将 `v` 和掩码 `mask` 进行按位或（将二者每个对应位上的值做或运算），将 `v` 的第 n 位置为 1，同时其它位不受影响，并返回结果。

4. 指定位置设置为0

问题描述：

将参数v的第n位置为0，然后返回该结果值。

答案：

```
public class Test4_SetZero {
  public static void main(String[] args) {
      //0b 0000 1010
      int r = setZero(10,2);
      System.out.println("r: " + r);
  }

  // 假设v=10    0b 0000 1010
  //    n=2     &1111 1101   <-  0000 0010 <-  1左移得到
  //            0000 1000 即可
  public static int setZero(int v,int n) {
        int mask = ~(1 << (n-1));
        int t = mask & v;
        return t;
  }
}
```

> 首先，左移操作 `1 << (n - 1)` 可以将数字 `1` 左移 `n - 1` 位，得到一个只有第 n 位为 1 的二进制数，名为掩码（mask）。例如当 n=3 时，掩码为 4 (二进制 100)。然后，`~(1 << (n - 1))` 表示按位取反该掩码，得到一个只有第 n 位为 0 的数，即需要将第 n 位设为 0 的掩码。
>
> 最后，使用按位与运算 `&` 将 `v` 和掩码 `mask` 进行按位与（将二者每个对应位上的值做与运算），将 `v` 的第 n 位置为 0，同时其它位不受影响，并返回结果。

流程控制

5. 代码分析

分析下面几段代码，写出输出结果，最后运行程序进行验证

1)

```java
int b=5;
if(b>4)
  System.out.println(b);
else
  b--;
System.out.println(b);
```

答案：

这段代码的执行流程如下：

1. 创建一个整型变量 `b` 并将其初始化为 `5`。
2. 检查 `b` 是否大于 `4`。由于 `b` 的值为 `5`，这个条件为真，因此控制流程跳转到下一行。
3. 执行 `System.out.println(b);` 语句，输出 `5`。
4. 控制流程跳转到代码块的结尾，继续执行下一行。
5. 输出 `5`。

因此，这段代码的输出将是：

```
5
5
```

> 由于 `b` 大于 `4`，所以第一个 `println` 语句被执行，输出 `5`。然后程序继续执行第二个 `println` 语句，输出 `5`。
>
> 在if...else....等流程控制语句中，如果没有使用{}对代码块进行包裹，那么在满足条件之后只会执行距离条件最近的一条语句

2)

```
1  {
2    int b=5;
3    if(b>4)b--; System.out.println("b大于4");
4    else {
5      System.out.println("b不大于4");
6    }
7   }
```

答案：

> 上述代码存在一个语法错误，在 if 和 else 语句块之间缺失了一对花括号，正确代码如下：

```
    int b = 5;
    if (b > 4) {
   b--;
   System.out.println("b大于4");
    } else {
   System.out.println("b不大于4");
    }
```

3)

```
int age=45;
if(age>20) {
    System.out.println("年轻人");
}else if(age>40){
    System.out.println("中年人");
}else if(age>60){
    System.out.println("老年人");
}
```

答案:

> 因为第一个 `if` 语句已经满足条件，所以后面的 `else if` 语句都不会执行。因此，输出结果将是 "年轻人"。

4)

```
```

```java
int a=80;
switch(a) {
  case 90:{
    System.out.println("优秀");
  }
  case 80:{
    System.out.println("一般");
  }
  case 60:{
    System.out.println("及格");
    break;
  }
  case 50:{
    System.out.println("基础弱");
  }
}
```

答案：

> 这段代码使用了 `switch` 语句，根据整型变量 `a` 的值输出不同的提示信息。
>
> 首先，`a` 的值为 `80`，控制流程进入 `switch` 语句，并执行第一个 `case` 分支。由于该分支没有使用 `break` 关键字，控制流会继续执行下一个 `case` 分支。

因此，程序将输出 "一般" 和 "及格" 两个字符串。注意，由于第三个 `case` 分支使用了 `break`，控制流程将跳出 `switch` 语句，不会执行最后一个 `case` 分支。

总结一下，`switch` 语句可以根据变量的值来选择执行不同的代码块。每个 `case` 分支代表一种取值，程序将会选择和变量匹配的 `case` 分支执行，从而达到不同的功能。需要注意的是，如果没有使用 `break` 关键字，控制流程将会穿过后面的所有分支，直到遇到 `break` 或 `switch` 语句的结束。因此，在编写 `switch` 语句时，需要格外小心，确保程序的执行逻辑是正确的。

运行结果如下：

```
一般
及格
```

5)

```
int count=0;
while(count>0);
{
    System.out.println(count);
}
```

答案：

上述代码有语法错误和逻辑错误，正确代码如下：

```
int count = 0;
while (count > 0) {
 System.out.println(count);
}
```

解释：

首先定义计数器变量 `count` 的值为 0。然后使用 `while` 循环语句对计数器变量 `count` 进行判断，只有当 `count` 大于 0 的时候才会进入循环体执行其中的代码。在本例中，由于 `count` 的初值为 0，因此不会进入循环体，直接跳过，程序结束。

原代码中，`while` 循环条件为 `count > 0`，而计数器变量 `count` 的初始值为 0，因此不符合循环条件，不会进入循环体。循环体内没有对计数器变量 `count` 进行修改，因此如果能够进入循环体，程序会无限循环下去，造成死循环和程序崩溃。

6)

```java
for (int i = 0; i < 5; i++) {
    System.out.print(i);
    i*=0.1;
}
System.out.println("循环结束");
```

答案：

1. 首先使用 `for` 循环语句初始化变量 `i` 的值为 0，设置循环条件为 `i < 5`，每次循环时 `i` 的值自增 1。
2. 在循环体内，使用 `System.out.print()` 方法打印 `i` 的值。第一次循环时，`i` 的值为 0，输出 0。
3. 将 `i` 乘以 0.1，即 `i *= 0.1`。由于 `i` 是整数，相乘得到的结果也是整数，因此小数部分会被舍去，`i` 的值变为 0。
4. 结束本次循环，将进入下一次循环，此时 `i` 的值已经被修改为 0，自增+1后`i`的值为1，由于循环条件 `i < 5` 仍然成立，因此会一直处于循环之中。
5. 由于最后出现死循环，因此程序不会输出 `"循环结束"`。

因此程序会输出一个 0后持续不断的输出1，并进入死循环

```
0111111111.....
```

7)

```
for(int i=0;i<3;i++) {
    System.out.println(i);
    if(i==1) {
        continue;
    }
    System.out.println("continue后的输出语句");
}
```

答案：

在这个循环中，变量 `i` 的初始值为 0。每次迭代开始时，首先输出 `i` 的值，然后判断 `i` 是否等于 1。当 `i` 等于 1 时，执行 `continue` 语句跳过当前循环的剩余部分，直接进入下一次迭代。因此，第二次迭代中的输出语句 `"continue后的输出语句"` 会被跳过，而其他的输出语句都会被执行。

需要注意的是，如果 `continue` 语句没有被执行，即当 `i` 不等于 1 时，循环体内的两个输出语句都会被执行。因此，循环体中的输出语句总数为 3 次，而不是 2 次。

代码输出结果如下：

```
0
continue后的输出语句
1
2
continue后的输出语句
```

8)

```java
for (int i = 0; i < 3; i++) {
    System.out.println(i);
    if (i == 1) {
        return;
    }
    System.out.println("return后的输出语句");
}
```

答案：

当 `i` 的值为 1 时，`continue` 语句会跳过后面的语句，直接进入下一次循环，因此输出的结果是 0 和 1。在第二次循环中，由于 `i` 的值为 1，`continue` 语句会跳过输出语句 `"continue后的输出语句"`，直接执行下一次循环，因此输出值为 2 的语句不会被执行。

执行结果如下：

```
0
return后的输出语句
1
```

6. 九九乘法表

问题描述：

编写一个Java程序，实现如下效果的九九乘法表

```
1*1=1
1*2=2   2*2=4
1*3=3   2*3=6   3*3=9
1*4=4   2*4=8   3*4=12  4*4=16
1*5=5   2*5=10  3*5=15  4*5=20  5*5=25
1*6=6   2*6=12  3*6=18  4*6=24  5*6=30  6*6=36
1*7=7   2*7=14  3*7=21  4*7=28  5*7=35  6*7=42  7*7=49
1*8=8   2*8=16  3*8=24  4*8=32  5*8=40  6*8=48  7*8=56  8*8=64
1*9=9   2*9=18  3*9=27  4*9=36  5*9=45  6*9=54  7*9=63  8*9=72  9*9=81
```

答案：

```java
public class Test6_MultiplicationTable {
    public static void main(String[] args) {
        for (int i = 1; i <= 9; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print(j + " × " + i + " = " + (i * j) + "\t");
            }
            System.out.println();
        }
    }
}
```

7. 素数问题

问题描述：

请编写一个Java程序，判断101-200之间有多少个素数，并输出所有素数

> 只能被1和它本身整除的自然数为素数 如101、5等

答案：

```java
public class Test7_PrimeNumber {
    public static void main(String[] args) {
        int count = 0;
        for (int i = 101; i <= 200; i++) {
            boolean isPrime = true;
            for (int j = 2; j <= Math.sqrt(i); j++) {
                if (i % j == 0) {
                    isPrime = false;
                    break;
                }
            }
            if (isPrime) {
                count++;
                System.out.print(i + " ");
            }
        }
        System.out.println("\n101-200之间共有 " + count + " 个素数");
    }
}
```

输出如下：

```
101 103 107 109 113 127 131 137 139 149 151
157 163 167 173 179 181 191 193 197 199
101-200之间共有 21 个素数
```

> 上述代码通过循环遍历 101-200 范围内的整数，对于每个整数判断是否为素数，如若是，输出，并记录素数的数量。在判断是否为素数时，遍历小于该整数的所有正整数，当整数能被其中之一整除时，说明该整数不是素数，因为素数只能被 1 和其本身整除。为了避免无谓的循环，我们判断到该整数的平方根即可，因为如果存在两个大于 1 的因数，其中一个必定小于或等于其平方根。

8.月份计算

问题描述：

请编写一个Java程序，实现从键盘录入月份，判断该月份属于哪个季节

【要求】

- 如果月份不再1-12之间，则报错：输入月份有误
- 匹配月份，输出对应的季节 12-2冬 3-5春 6-8夏 9-11秋

- 使用`java.util.Scanner`类完成

答案：

```
1  import java.util.Scanner;
2
3  public class Test8_Season {
4      public static void main(String[] args) {
5          Scanner sc = new Scanner(System.in);
6          System.out.print("请输入一个月份：");
7          int month = sc.nextInt();
8          String season;
9          switch (month) {
10             case 12:
11             case 1:
12             case 2:
13                 season = "冬季";
14                 break;
15             case 3:
16             case 4:
17             case 5:
18                 season = "春季";
19                 break;
20             case 6:
21             case 7:
22             case 8:
```

```
23                      season = "夏季";
24                      break;
25                  case 9:
26                  case 10:
27                  case 11:
28                      season = "秋季";
29                      break;
30                  default:
31                      season = "输入月份有误";
32                      break;
33              }
34              System.out.println(month + " 月份属
    于: " + season);
35          }
36      }
```

程序中使用了 `Scanner` 类从控制台读取用户输入的月份。然后，根据月份使用 `switch` 语句判断该月份属于哪个季节，并将结果输出到控制台。注意，此处使用了 `case` 中穿透的技巧，即多个 `case` 共用一个分支语句，避免了重复代码。

9. 天数计算

问题描述：

请编写一个Java程序，实现从键盘中输入年份 月份 日期，输出这一天是该年的第几天

【要求】

- 输入的年份、月份、日期均为数字，不足10的用0x表示，如：1月1日为01 01
- 输入的年份、月份、日期之间用空格隔开，如：2023 04 27

【示例】

输入：2023 04 27

输出：117

> 如 输入 2023 04 27
>
> 输出 117

答案：

```
1  import java.util.Scanner;
2
3  public class Test9_DayOfYear {
```

```
4       public static void main(String[] args) {
5           Scanner scan = new
    Scanner(System.in);
6           System.out.print("请输入年份: ");
7           int year = scan.nextInt();
8           System.out.print("请输入月份: ");
9           int month = scan.nextInt();
10          System.out.print("请输入日期: ");
11          int day = scan.nextInt();
12          scan.close();
13
14          int dayOfYear = 0;
15          for (int i = 1; i <= month; i++) {
16              if (i == 2) {
17                  if (year % 400 == 0 || (year
    % 4 == 0 && year % 100 != 0)) {
18                      dayOfYear += 29;
19                  } else {
20                      dayOfYear += 28;
21                  }
22              } else if (i == 4 || i == 6 || i
    == 9 || i == 11){
23                  dayOfYear += 30;
24              } else {
25                  dayOfYear += 31;
26              }
27          }
```

```
28            dayOfYear += day;
29            System.out.println(year + "年" +
    month + "月" + day + "日是该年的第" +
    dayOfYear + "天");
30        }
31    }
```

> 该程序会根据输入的年份、月份和日期计算出当日是该年的第几天，并输出结果。其中 for 循环遍历到输入的月份，根据不同的月份来计算每个月的天数。闰年的计算采用了普通闰年、世纪闰年以及400年周期的规则。

方式二：

以下是不使用数组或集合的 Java 程序实现计算输入日期是该年的第几天：

```
1   public class Test9_DayOfYear_2 {
2     public static void main(String[] args) {
3           Scanner sc = new Scanner(System.in);
4           System.out.print("请输入年份: ");
5           int year = sc.nextInt();
6           System.out.print("请输入月份: ");
7           int month = sc.nextInt();
```

```
 8            System.out.print("请输入日期: ");
 9            int day = sc.nextInt();
10            sc.close();
11            int days = 0;
12            // 计算当年之前所有月份的天数
13            for (int i = 1; i < month; i++) {
14                switch (i) {
15                    case 1:
16                    case 3:
17                    case 5:
18                    case 7:
19                    case 8:
20                    case 10:
21                    case 12:
22                        days += 31;
23                        break;
24                    case 2:
25                        days += isLeapYear(year)
   ? 29 : 28;
26                        break;
27                    default:
28                        days += 30;
29                        break;
30                }
31            }
32            // 加上当月的天数
33            days += day;
```

```
34          System.out.println(year + " 年 " + month + " 月 " + day + " 日是该年的第 " + days + " 天");
35      }
36
37      // 判断是否为闰年
38      private static boolean isLeapYear(int year) {
39          return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
40      }
41  }
```

10. 完全数问题

问题描述：

请编写一个Java程序，实现输入一个数，判断其是否为完全数

> 若一个自然数，恰好与除去它本身以外的一切因数的和相等，这种数叫做完全数。
>
> 如：

> 6是完全数，因为，6 = 1 + 2 + 3;
> 28是完全数，因为，28 = 1 + 2 + 4 + 7 + 14;

【示例1】
输入：6

输出：6是完全数

【示例2】
输入：11

输出：11不是完全数

答案：

```
1  import java.util.Scanner;
2
3  public class Test10_PerfectNumber  {
4      public static void main(String[] args) {
5          Scanner scan = new
   Scanner(System.in);
6          System.out.print("请输入一个正整数: ");
7          int num = scan.nextInt();
8          scan.close();
9
10         int sum = 0;
11         for (int i = 1; i < num; i++) {
```

```
12                 if (num % i == 0) {
13                     // 如果i是num的因数，则加到和
    sum中
14                     sum += i;
15                 }
16             }
17             if (sum == num) {
18                 System.out.println(num + "是一个
    完全数");
19             } else {
20                 System.out.println(num + "不是一
    个完全数");
21             }
22         }
23     }
```

> 程序使用 for 循环从1到num-1逐一判断num的因子，并将因子加入sum中。当循环结束后，如果sum等于num，则说明num是完全数。如果不等于，则说明num不是完全数。

11. 图形输出

问题描述：

请编写一个Java程序，实现下面图形的输出

```
    *
   ***
  *****
 *******
  *****
   ***
    *
```

【提示】

- 先输出上半部分【先输出空格 再输出*】
- 再输出下半部分

答案：

```
public class Test11_PrintDiamond {
    public static void main(String[] args) {
        //输入菱形的高度
        int height = 7;
        //输出上半部分
        for (int i = 1; i <= height / 2 + 1; i++) {
            //打印空格
```

```
 8                for (int j = 0; j < height / 2 +
    1 - i; j++) {
 9                    System.out.print(" ");
10                }
11                //打印*
12                for (int k = 1; k <= 2 * i - 1;
    k++) {
13                    System.out.print("*");
14                }
15                //换行
16                System.out.println();
17            }
18            //输出下半部分
19            for (int i = height / 2; i >= 1; i--
    ) {
20                //打印空格
21                for (int j = 0; j < height / 2 +
    1 - i; j++) {
22                    System.out.print(" ");
23                }
24                //打印*
25                for (int k = 1; k <= 2 * i - 1;
    k++) {
26                    System.out.print("*");
27                }
28                //换行
29                System.out.println();
```

```
30            }
31        }
32    }
```

> 这个程序的思路是先输出菱形的上半部分，再输出菱形的下半部分。在输出菱形的上半部分时，每行的空格数是递减的，每行的星号数是递增的；在输出菱形的下半部分时，每行的空格数是递增的，每行的星号是递减的

12. 分解质因数

问题描述：

请编写一个Java程序，实现输入一个15以内的正整数，对该正整数进行分解质因数，输出该整数与质因数的等式。

【示例1】

输入：3

输出：3 = 3

【示例2】

输入：20

输出：输入数字不符合要求

【示例3】

输入：8

输出：8 = 2 * 2 * 2

答案：

```
import java.util.Scanner;

public class Test12_PrimeFactorization {
    public static void main(String[] args) {
        Scanner scanner = new
Scanner(System.in);
        System.out.print("请输入一个15以内的正
整数: ");
        int num = scanner.nextInt();
        System.out.print(num + "=");
        for (int i = 2; i <= num; i++) {
            while (num % i == 0) {
                System.out.print(i);
                num /= i;
```

```
13                     if (num != 1) {
14                         System.out.print("*");
15                     }
16                 }
17             }
18         }
19     }
```

首先，我们引入了Scanner类来接收用户输入的正整数。接下来，我们用一个for循环来遍历从2到该正整数本身的所有数字，这些数字都有可能是该数的因子。

在循环中，如果该数字能够整除该正整数，说明它是该正整数的一个因子。此时，我们打印出该因子，并且将该正整数除以该因子，以便继续寻找该正整数的其他因子。如果该正整数不能整除该数字，说明该数字不是该正整数的因子，我们就继续往下一个数字遍历。这个过程中，我们通过while循环来将一个因子拆分为多个相同因子的乘积。

最后，当该正整数被全部分解为1时，我们就结束了整个因数分解的过程，输出结果。