前备知识：二进制数的算术运算

1. 无符号二进制数的算术运算：

- 二进制加法：

```
0 + 0 = 0，
0 + 1 = 1，
1 + 1 = 10
```

- 二进制减法：

```
0 - 0 = 0，
1 - 1 = 0，
1 - 0 = 1，
0 - 1 = 11
```

- 乘法运算和除法运算：

  乘法：

```
0 * 0 = 0，
0 * 1 = 0，
1 * 0 = 0，
1 * 1 = 1
```

  除法：

```
0 / 0 = 0，
0 / 1 = 0，
1 / 0（无意义，不存在），
1 / 1 = 1
```

2. 带符号二进制数的减法运算：

在定点运算的情况下，二进制数的最高位表示符号位，0表示正数，1表示负数。如：

$$(+11)_D = (01011)_B$$
$$(-11)_D = (11011)_B$$

- 二进制的补码表示:

  计算机中的所有数都以补码形式存在,以便将减法运算变为加法运算

  带符号的二进制数补码计算方法如下:

  1. 补码或反码的最高位为符号位, 正数为0, 负数为1
  2. **当二进制数为正数时, 其补码.反码与原码相同**
  3. 当二进制为负数时,**将原码的数值位逐位求反(即得到反码), 然后在最低位加*1*得到补码.**

     (注:负数反码在最低位加1相当于反码得到的数减1)

     原码.反码和补码的范围分别是:

类型	范围
原码	$-(2^{n-1}-1) \sim +(2^{n-1}-1)$
反码	$-(2^{n-1}-1) \sim +(2^{n-1}-1)$
补码	$-2^{n-1} \sim +(2^{n-1}-1)$

- 二进制补码的减法运算

  进行二进制补码加法运算时, 必须注意被加数补码与加数补码的位数相等,即让两个二进制补码的符号位对其.

  例: 5 - 2:

```
解:  (5 - 2)补 = (5)补 + (-2)补
                = 0101 + 1110
                = 0011

注: 0101 + 1110 = 10011, 最高位1自动丢弃
```

  例: 5 + 7:

```
解: (5 + 7)补 = (5)补 + (7)补
              = 0101 + 0111
              = 1100
              = -4(结果明显错误)

错误原因: 4位二进制补码中, 3位表示数值位, 因此表示范围为 -8 ~ +7, 而5 + 7 = 12 > +7, 因此产生溢出
```

- 溢出的判别
  1. 两个符号相反的数相加不会产生溢出
  2. 两个符号相同的数相加:

$$\begin{array}{r} 1101 \\ +1010 \\ \hline [1]0111 \end{array}$$

当方框中的进位位与计算结果的符号位相反时,则运算结果是错误的,产生溢出(上式中的[1]和0)

位运算

1. 基本操作

(以下运算皆在二进制下进行)

运算符	含义	实例	结果
>>	右移	4 >> 1	2
<<	左移	4 << 2	16
>>>	无符号右移	-24 >>> 1	1073741818
&	与	4 & 2	0
|	或	4 | 2	6

2. 左移右移

- **左移**:

```
操作数 << 位数
```

将符号左边的操作数左移指定的位数, 左边最高位丢弃,右边补0

```
如:-7 << 1

    -7的二进制:
    原码: 10000000 00000000 00000000 00000111
    反码: 11111111 11111111 11111111 11111000
    补码: 11111111 11111111 11111111 11111001
    左移1位，左边最高位丢弃，右边补齐0:
    左移后的补码: 11111111 11111111 11111111 11110010
    原码: 10000000 00000000 00000000 00001110 (-14)
```

注: 左移1位即左边数乘以2的1次方，左移2位即左边数乘以2的2次方，以此类推

- **右移**:

```
操作数 >> 位数
```

将符号左边的操作数右移指定的位数, 最高位是0, 左边补齐0, 最高为是1, 左边补齐1。

```
如:24>>2

    24的二进制:
    补码: 00000000 00000000 00000000 00011000
    右移2位，最高位是0，左边补齐0；最高为是1，左边补齐1:
    右移后的补码(原码): 00000000 00000000 00000000
00000110
```

注: 右移1位即左边数除以2的1次方，右移2位即左边数除以2的2次方，以此类推

- **无符号右移**(没有无符号左移):

```
操作数>>>拉数
```

将符号左边的操作数右移指定的位数，无论最高位是0还是1，左边补0

```
如:-24 >>> 2

    -24的二进制:
    原码: 10000000 00000000 00000000 00011000
    反码: 11111111 11111111 11111111 11100111
    补码: 11111111 11111111 11111111 11101000
    右移2位,左边补0
    结果: 0011111111 11111111 11111111 111010
(1073741818)
    (注意，如果最高位1，则还需要求出原码)
```

操作使用

1. 奇偶判断:

```java
public class Deom {
    //运用位运算来进行奇偶判断
    public static void main(String args[]) {

        int a = 1;
        int x = 6;
        int y = 7;
        //和1与运算，结果为1则为奇数
        System.out.println(x & a);
        //和1与运算，结果为0则为偶数
        System.out.println(y & a);

    }
}
```

2. 两数交换:

```java
public class Deom {
    // 运用位运算来进行两数交换
    public static void main(String args[]) {
        // 一个数与自身异或为0,再与另一个数异或则变成另一个数
        int a = 5;
```

```
        int b = 10;

        // a ^ a ^ b = b
        a = a ^ b;
        b = a ^ b;
        a = b ^ a;

        System.out.println(a);//10
        System.out.println(b);//5
    }
}
```