复习：

1. 注释：是程序中用于解释说明的文字 - 文档注释中的文字可以利用javadoc命令进行提取

2. 字面量：整数、小数、字符、字符串、布尔、空

3. 进制：实际上是一种计数方式。

二进制：0-1，满2进1，需要以0b开头（从JDK1.7开始）

八进制：0-7，满8进1，需要以0开头

十进制：0-9，满十进1，程序中默认就是十进制

十六进制：0-9，a-f，满16进1，需要以0x开头

注意进制之间的转换规则

4. 变量：数据类型 变量名 = 数据; - 变量必须在定义的范围内使用；变量必须先定义后使用，也必须先给值后使用

5. 数据类型：

基本数据类型

byte short int long float double char boolean

byte的取值范围：-128~127

整数默认为int，小数默认为double

long类型需要以l/L作为结尾

float类型需要以F/f作为结尾

char类型的默认编码是utf-16 - 2个字节

boolean用于表示逻辑值

引用数据类型

数据[] 类class 接口interface

6. 数据类型的转换

隐式转换

小类型可以自动转化为大类型

整数可以自动转化为小数，可能会产生精度损失

字符可以自动转化为整数

显式转换

大类型转换为小类型，因为字节的损失可能导致数据不精确

小数转换为整数的时候需要舍弃小数位

## 运算符

### 算术运算符

+ - \* / % ++ --

注意：

1. byte/short/char在运算的时候会自动提升为int

2. 整数的运算结果一定是整数

3. 小类型和大类型运算结果一定是大类型

4. 任意整数/0 - ArithmeticException

非零数字/0.0 非零小数/0 - Infinity

0.0/0.0 0.0/0 0/0.0 - NaN - 非数字

% 取模运算 - 取余运算

9%4=1 3%8=3

-9%4=-1 9%-4=1 -9%-4=-1

%的结果的正负看的是%左边数据的正负 -> 如果左边的数字是一个正数，那么结果就是正数；反之，那么结果就是负数

5%1.4=0.8 6.3%1.7=1.2

++/-- 自增/自减运算

++/--如果在变量之前，那么先自增/自减，然后参与后续运算

++/--如果在变量之后，那么先将值取出来参与运算，然后变量再自增/自减

int i = 6;

// int j = i++ + ++i; -> 14

int j = ++i + i++; -> 14

byte/short/char都可以参与自增运算

char c = ‘b’;

int i = c + 2; -> 100

‘a’ ~ ‘z’ -> 97~122 ‘A’ ~ ‘Z’ -> 65~90 ‘0’ ~ ‘9’ -> 48~57

char c = ‘0’;

int i = c + 3; -> 51

### 赋值运算符

= += -= \*= /= %= &= |= ^= <<= >>= >>>=

byte/short/char可以参与赋值运算

除了=以外，其余的符号都要求这个变量得先有值

注意:在Java中不支持连等定义但是支持连等赋值

int i = 5;

i += i \*= i -= 3; -> 15

i = 5 + (5 \* (5 - 3));

int j = 7;

j -= j \*= j++; -> -42

j = 7 - (7 \* 7);

### 关系/比较运算符

==相等 !=不等 > < >= <=

注意：不支持连着的比较方式

### 逻辑运算符

注意：逻辑运算符算的是逻辑值

&与 |或 !非 ^异或 &&短路与 ||短路或

true&true=true true&false=false false&true=false

false&false=false

true|true=true true|false=true false|true=true

false|false=false

!true=false !false=true

true^true=false true^false=true false^true=true

false^false=false

&&：如果前边的表达式的结果为false，则后边的表达式不再运算，整个表达式的结果就确定为false

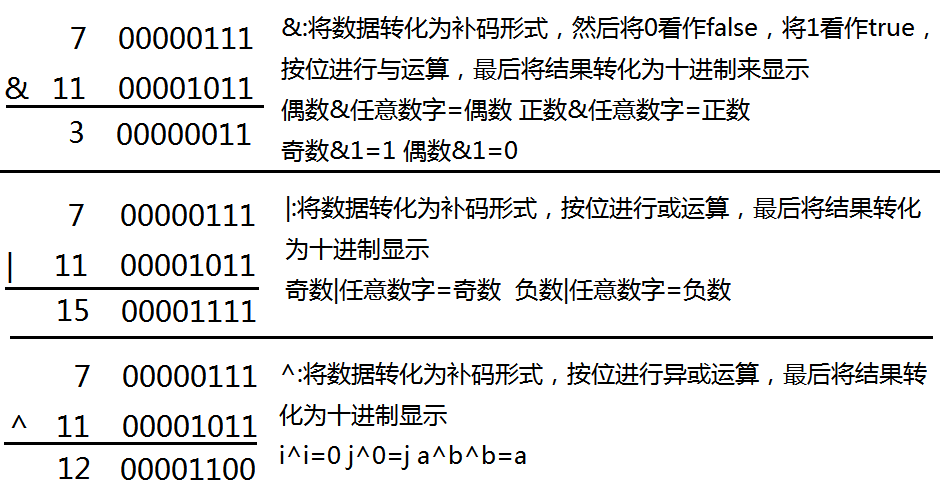
||：如果前边的表达式的结果为true，则后边的表达式就不再运算，整个表达式的结果就确定为true

注意：||如果在&&的前边可以把&&短路掉

### 位运算符

注意：位运算符针对整数的补码进行运算，所以运算结果也是补码

& | ^ << >> >>> ~取反



#### 交换值的方式

方式一：异或法

int i = 5, j = 9;

i = i ^ j;

j = i ^ j; -> j = i ^ j ^ j; -> j = i;

i = i ^ j; -> i = i ^ j ^ i; -> i = j;

方式二：加减法

int i =5, j = 9;

i = i + j;

j = i - j; -> j = i + j - j; -> j = i;

i = i - j; -> i = i + j - i; -> i = j;

方式三：追尾法

int i = 5, j = 9;

int temp = i;

i = j;

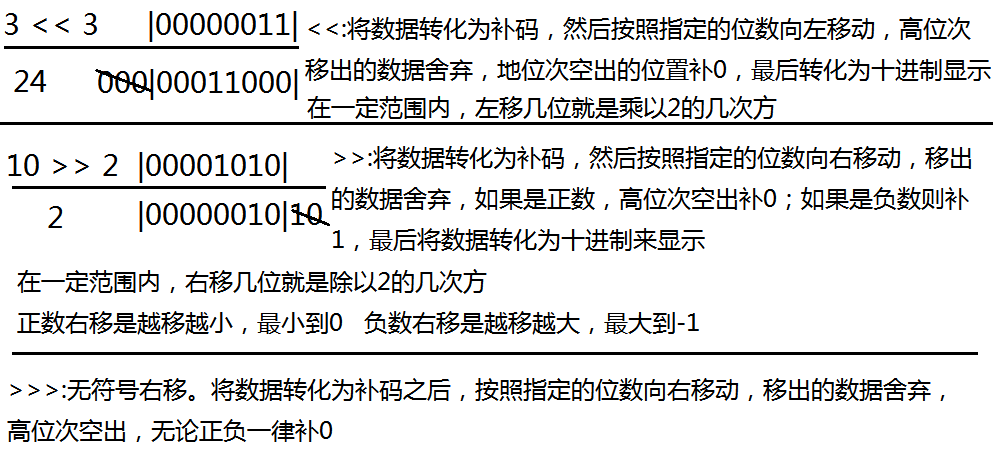
j = temp;

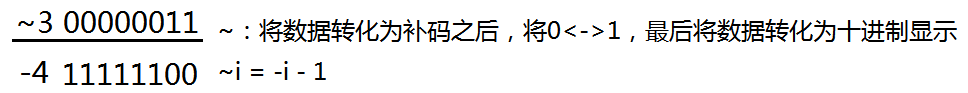
总结：三种方式的比较

异或法的效率是最高的。只能针对整数进行交换，局限性太大

加减法的效率是低于异或但是高于追尾。理论上可以交换数值类型，但实际上很少用于交换小数

追尾法的效率是最低的。可以交换任意一个类型的值





### 三元运算符

格式：逻辑值 ? 表达式1 : 表达式2

执行顺序：先执行逻辑值，如果逻辑值为true，则执行表达式1；反之则执行表达式2

三元表达式本身是一个表达式，意味着这个表达式执行完成之后需要有一个结果 --- 这个结果必须能定义对应类型的变量来接住

a > b ? “abc” : true; -> 没有办法定义一个统一类型的结果来存储，这种写法就是错误的

double d = a > b ? 3 : 5.25;

float f = a > b ? ‘a’ : 3.5f;

练习： 输出分数对应的等级 >=90-A >=80-B >=70-C >=60-D <60-E

扩展：从控制台获取数据

import java.util.Scanner;

Scanner s = new Scanner(System.in);

int i = s.nextInt(); // 获取整数

double d = s.nextDouble(); // 获取小数

String str = s.next(); // 获取字符串

### 运算符的优先级

~ ! ++ -- \* / % + - << >> >>> 关系 逻辑 & | ^ 三元 赋值

一元 > 二元 > 三元 > 赋值

## 流程控制

顺序结构：指代码是从上到下从左到右来依次编译运行的

### 分支结构

#### 判断结构

if(逻辑值){

代码块;

}

执行顺序：先执行逻辑值，如果逻辑值为true，则执行代码块；反之则不执行

注意：如果if中的代码块只有1句话，那么可以省略{}不写

if(逻辑值){

Code1;

} else {

Code2;

}

执行顺序：先执行逻辑值，如果逻辑值为true，那么执行Code1；反之执行Code2

练习：

1. 输入三个数字，获取三个数字中的最小值

2. 输入一个数字表示重量，如果重量<=20，则每千克收费0.35元；如果超过20千克不超过100千克的范围，则超过的部分按照每千克0.5元收费；如果超过100千克，则超过的范围按照每千克0.8元收费。计算输入的重量对应的总费用

if(逻辑值1){

Code1;

} else if(逻辑值2){]

Code2;

}

...

else {

Code;

}

执行顺序：先执行逻辑值1，如果逻辑值1为true，则执行Code1；反之则执行逻辑值2，如果逻辑值2为true，则执行Code2；反之则顺次继续往下执行

练习：

输入一个数字表示月份，然后输出这个月份所对应的季节。3-5-春 6-8-夏 9-11-秋 12、1、2-冬