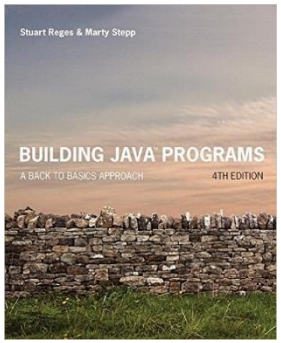
Java Statements

Reading Assignment: Read Chapters 2 and 4 from Building Java Programs (Stuart Reges and Marty Stepp)



Overview

- Java Statement
- Expression
- Data Conversion
- Compound statements; Alternative statements: if, switch; Repetitive statements: for, while, do/do-while

Java Statements

Java **control structure** is entrenched in statements:

- Simple statements
- Compound statements
- Alternative statements: if, switch
- Repetitive statements: for, while, do/do-while

Simple Statements

expression;

A simple statement requires a statement terminator.

Expressions

- What is an expression?

  "Any combination of operands and operators, which, when evaluated, yields a value"

- Example: num = 3.0 + (2.1 - x) * y / sqrt(v);
- What are operands?
  - Literals/constants [represent values]
  - Variables [contain values]
  - Function invocations [produce values]
  - Sub-expressions [results of evaluations]

Three characteristics of operators

- Arity: number of operands required
  - Unary, binary
- Precedence: determines the order in which operations are performed.
- Associativity: determines whether operations should occur left-right or right-left.

Operator precedence & associativity

Operator	Associativity	Precedence
() [] .	Left Assoc.	(High)
! ~ ++ -- + - (Data Type)	Right Assoc.	
* / %	Left Assoc.	
+ -	Left Assoc.	
<< >> >>>	Left Assoc.	
< <= > >=	Non-assoc.	
== !=	Non-assoc.	
&	Left Assoc.	
^	Left Assoc.	
|	Left Assoc.	
&&	Left Assoc.	
||	Left Assoc.	
? :	Left Assoc.	
= += -= *= /= %= &= ^= |= etc.	Right Assoc.	(Low)

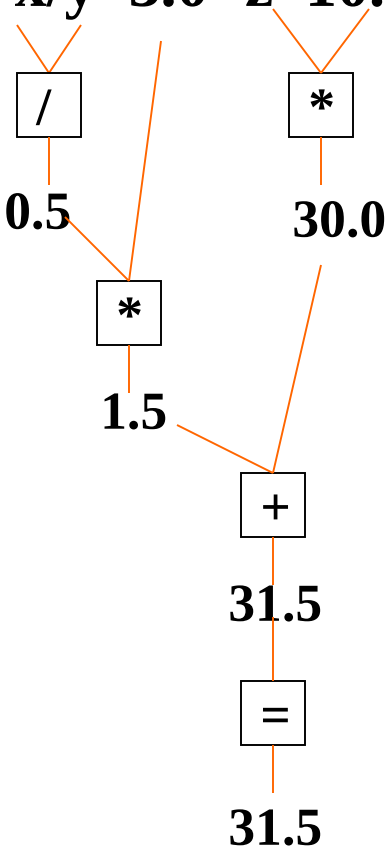
Example:

x = 1.0;

y = 2.0;

z = 3.0;

w = x/y*3.0+z*10.0;



Example:

```
x = 1;
y = 2;
z = 3;
x  += y *= z  = 4;
System.out.printf("x=%d, y=%d, z=%d\n", x, y, z);
```

x=9, y=8, z=4

Example:

```
x = 1;
y = 2;
z = 3;
z = x++ + ++y;
System.out.printf("x=%d, y=%d, z=%d\n", x, y, z);
```

```
x=2, y=3, z=4
```

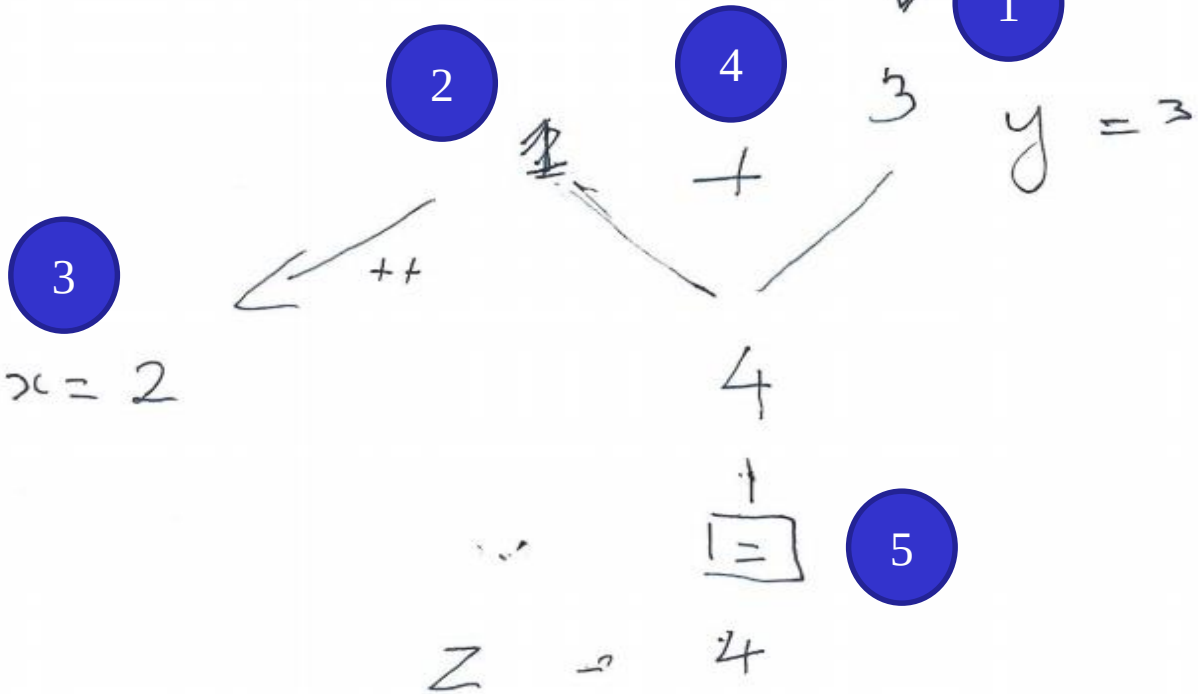
$Z = x++ + ++y;$

Row 2 Associative

and precedence!

higher than

=



x=2, y=3, z=4

Quiz

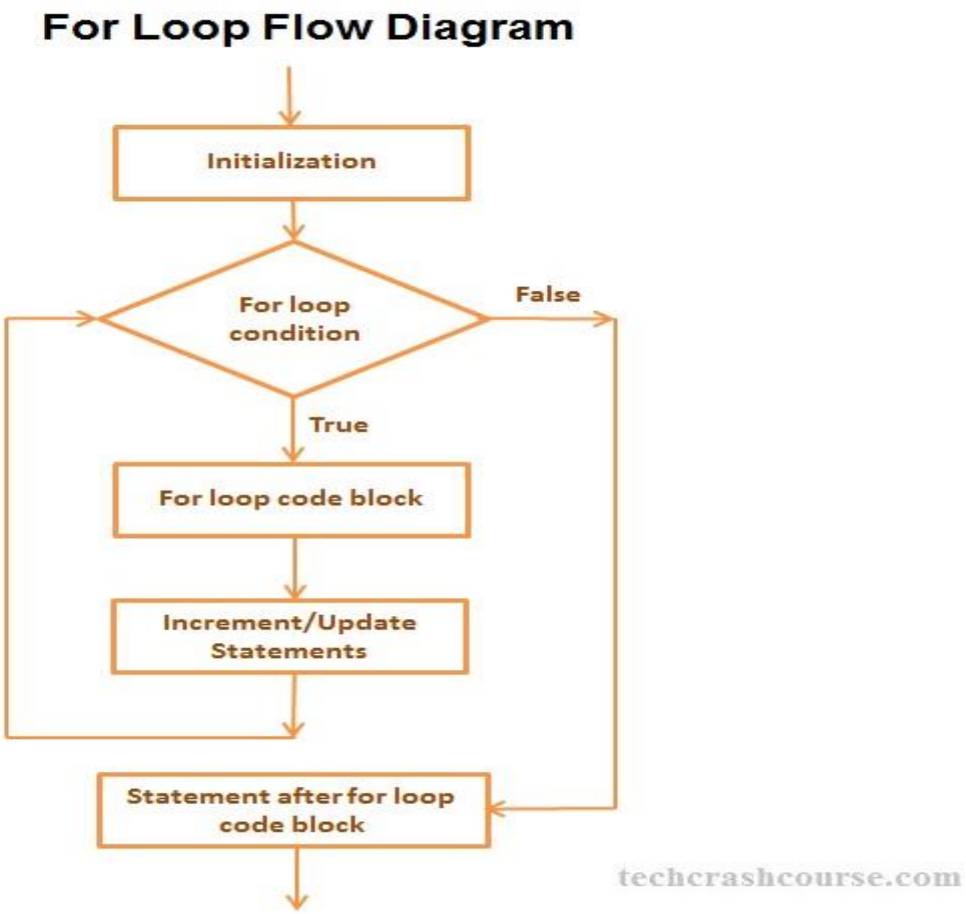
What is the difference between the two loops?

```
for (int x=0; x<10; ++x) {
    …
}
```

```
for (int x=0; x<10; x++) {
    …
}
```

For Loop Control Flow

```
for (int x=0; x<10; ++x) {
    …
}
```



Data Conversions

- In Java, when performing an arithmetic operation, all values must be of the same type.
- Sometimes it is necessary to convert data from one type to another.
- Conversions must be handled carefully to avoid losing information.
- *Widening conversions* are safest because they tend to go from a small data type to a larger one (such as a short to an int)
- *Narrowing conversions* can lose information because they tend to go from a large data type to a smaller one (such as an int to a short)

Data Conversions conti.

- In Java, data conversions can occur in three ways:
  - assignment conversion
  - arithmetic promotion
  - casting
- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
- *Arithmetic promotion* happens automatically when operators in expressions convert their operands
- Only widening conversions can happen for the above two cases.

Data Conversions conti.

- *Casting* is the most powerful, and dangerous technique for conversion.
- Both widening and narrowing conversions can be accomplished by explicitly casting a value
- To cast, the type is put in parentheses in front of the value being converted.
- For example, if total and count are integers, but we want a floating point result when dividing them, we can cast total:

```
result = (float) total / count;
```

Short-circuit Evaluation of Boolean Expressions

- In Java, boolean expressions are evaluated left to right until their truth value becomes known **then evaluation stops.**
- Example, To prevent divide by 0

  `(x != 0) && (y/x > 5)` is ok,

  while

  `(y/x > 5) && (x != 0)` is not (why not?)

Compound Statements

- A sequence of zero or more statements contained between { and }.

- Format:

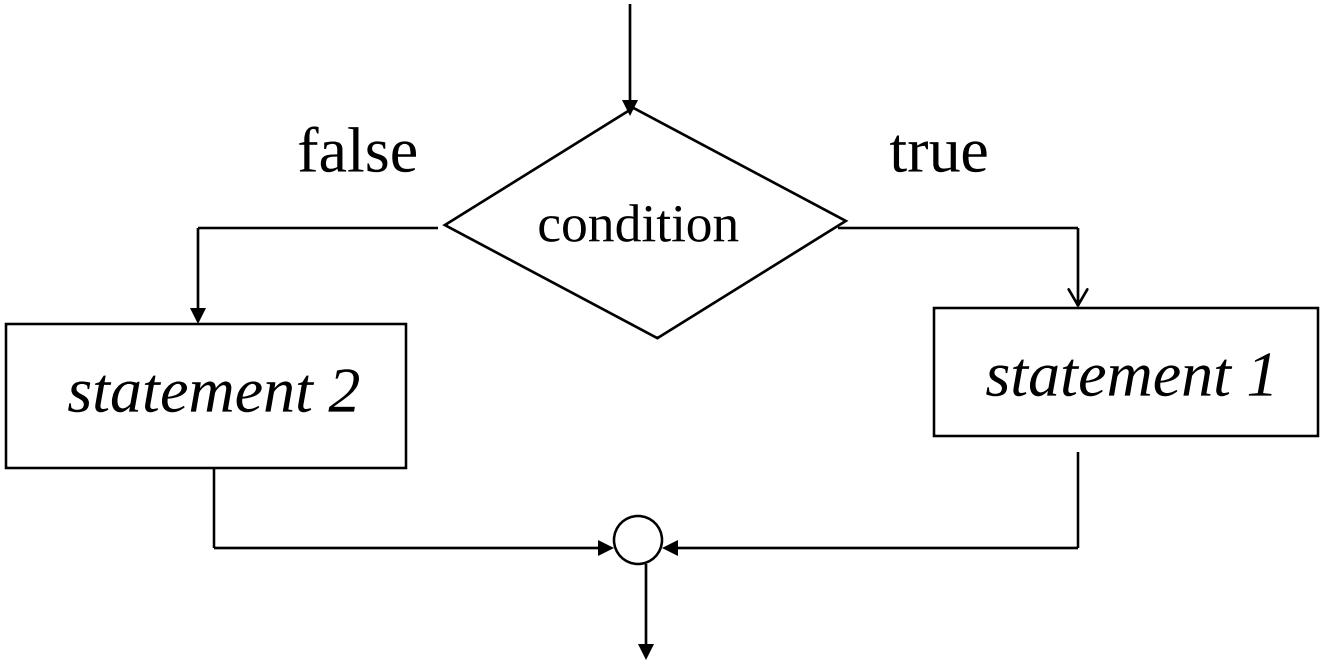
```
{   s1; s2; ….
}
```

- A compound statement is also called a **block**.
- A compound statement is considered as a single statement.

If Statements

Two forms:

- if (condition) statement
- if (condition) statement1 else statement2



A grammatical ambiguity in if

```
x = -1;
y = -1;
if (x>0) if (y>0) System.out.println("A"); else System.out.println("B");
```

```
x = -1;
y = -1;
if (x>0)
        if (y>0) System.out.println("A");
else System.out.println("B");
```

Indentation doesn't Help !!!

Result ?

The conditional operator

- Format:

  expr1 ? expr2 : expr3

- Examples:

  x = y>0 ? z*y : z+x;

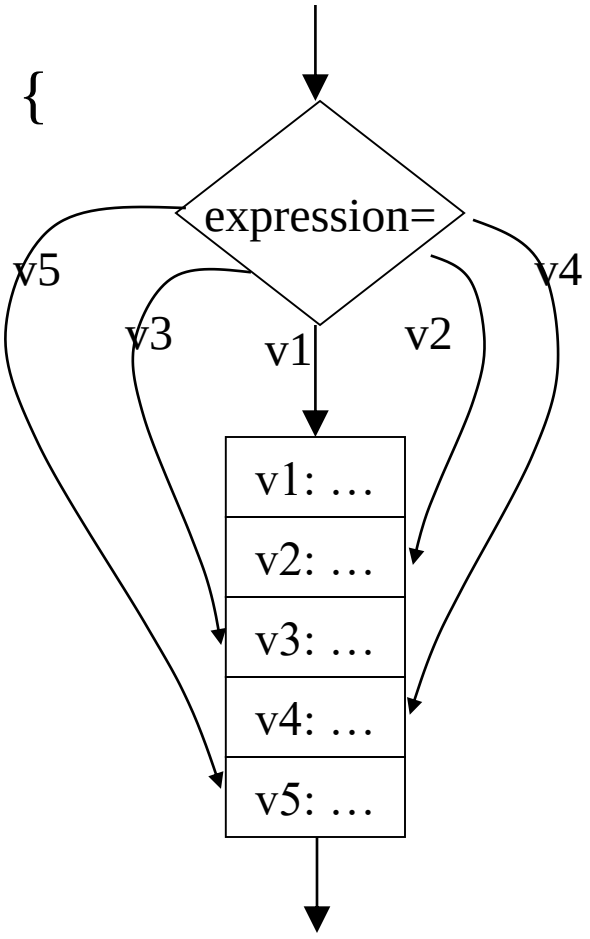
  Equivalent to :

  if (y>0) x = z*y; else x = z+x;

The Switch Statement

The format is:

```
switch (byte/char/short/int valued expression) {
  case value1: 0 or more statements;
  case value2: 0 or more statements;
  case value3: 0 or more statements;
  .
  default:  0 or more statements;
}
```



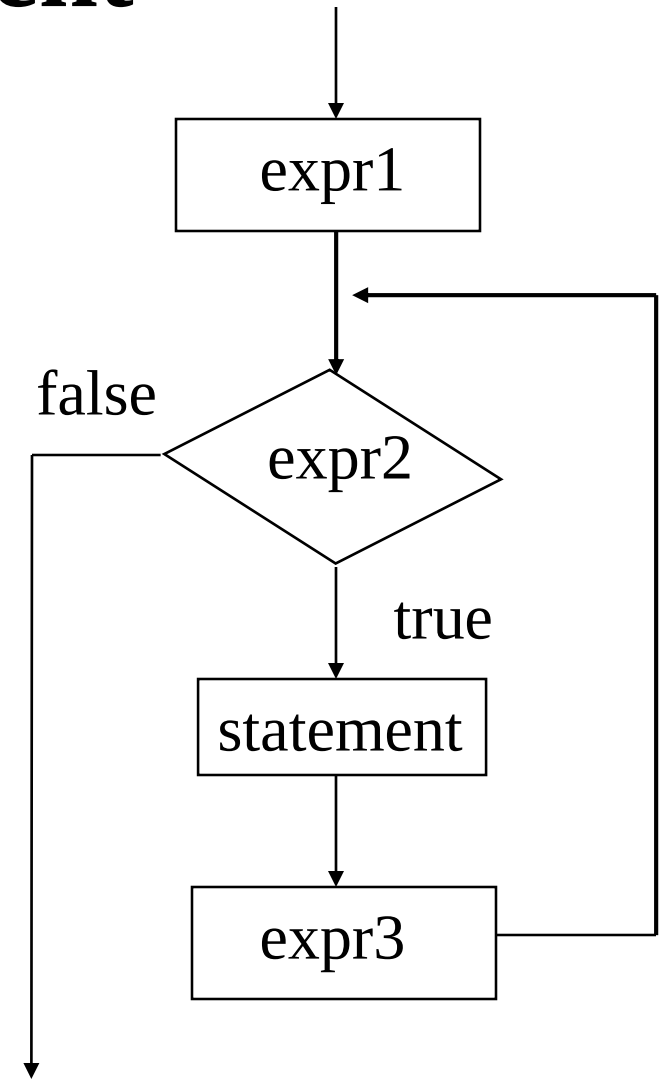
A switch example

```
switch (score/10) {
    case 10 :
    case 9 :   grade = ‘A’;  break;
    case 8 :   grade = ‘B’;  break;
    case 7 :   grade = ‘C’;  break;
    case 6 :   grade = ‘D’;  break;
    default :  grade = ‘F’;
}
```

The for statement

Format:

```
for (expr1; expr2; expr3)
    statement
```




```
for (count=1; count<=limit; count++) {
    int i = stdin.nextInt();
    sum += i;
}
```

The infinite for statement

```
for (;;) {
    statements
}
```

```
for (;;) {
    statements

    if (ExitCondition) break;

    statements
}
```

The comma operator

- Format:

  $expr_1, expr_2, expr_3, \ldots expr_n$

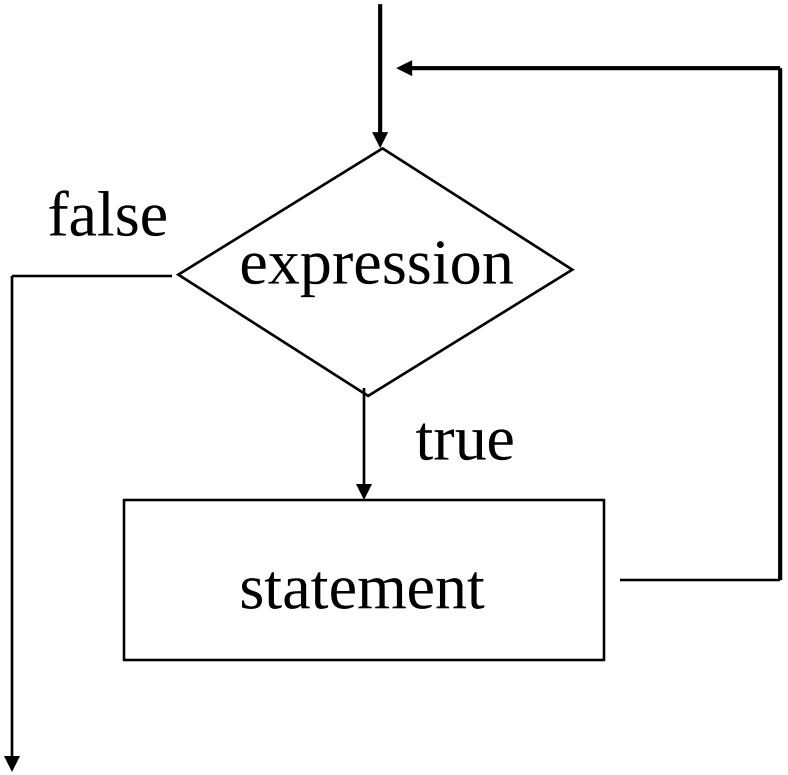
- Example:

```
for (i=0, j=1, k=2; i<10; ++i, ++j)
…
```

The while statement

```
while (expression) statement
```

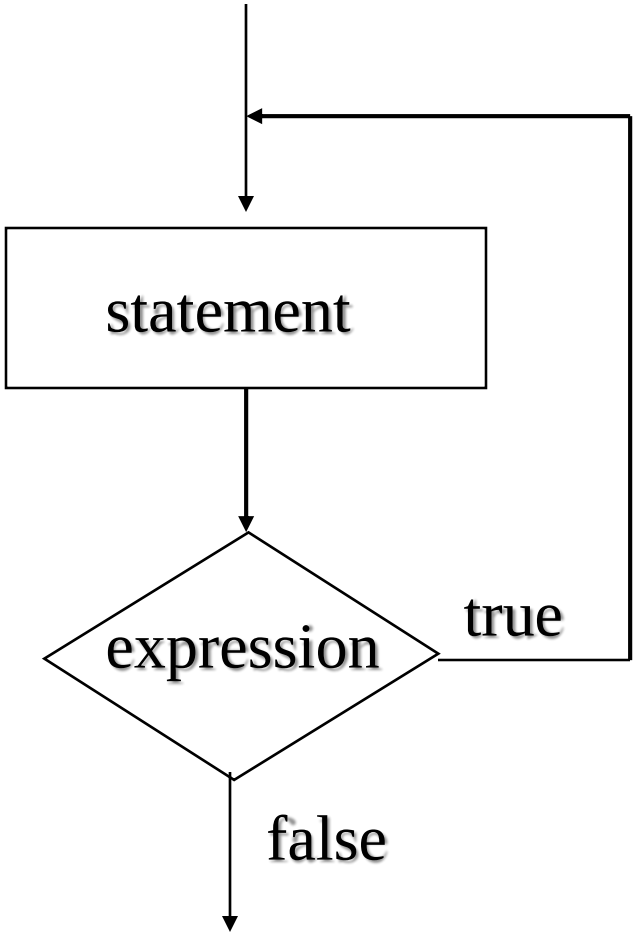
```
count = 1;
limit = 10;
sum = 0;
while (count <= limit) {
      int i = stdin.nextInt();
     sum += i;
     count++;
}
```



The do-while statement

Format:

```
do {
    statements
  } while ( expression );
```




```
count = 0;
do {
    int i = stdin.nextInt();
    sum += i;
    count++;
} while (count<=10);
```

The break statement

- Format:

  break;

  or

  break label;

- To force an abrupt termination or exit from a loop or a switch.

Break example

```
outer: while {
    inner: while {
                switch {
                        break; / break outer; / break inner;
                }
            }
        }
```

The continue statement

- Format:

  continue;

  or

  continue label;

- Similar to the break statement, but it just skips one loop iteration.
- In a while loop, jump to the test expression.
- In a do while loop, jump to the test expression.
- In a for loop, jump to the expression 3.

Continue example

```
outer: while {
    inner: while {
                switch {
                        continue; / continue outer;
                }
            }
        }
```

Homework # 2

1. Assume that integer variables x, y, and z contains 1, 2, and 3 respectively. Show lines printed.

   (a) if (x == y--) z += 3;

   System.out.printf("%d %d %d\n",x,y,z);

   (b) z = (int)(x/y*3.0+z*012);

   System.out.printf("%d %d %d\n",x,y,z);

   (c) x += y += z = 1;

   System.out.printf("%d %d %d\n",x,y,z);

Homework # 2

2. Show the output of the following loop.

```
for(i=1,j=0;i<=5;++i) {
   switch(i) {
    case 1: ++j;
    case 2: j *=2;
    case 3: break;
    case 4: continue;
    case 5: j--;
   }
   System.out.printf("i=%d,j=%d\n",i,j);
}
```