



Loops

while, do ... while, for loops
and how to use them

by James Brucker

Syntax of three loop statements

Loops containing a single statement:

```
while ( boolean_expression ) statement;
```

```
do statement; while ( boolean_expression );
```

```
for ( initializer ; boolean_expr ; increment ) statement;
```

The *statement* can be null, but semi-colon is required.

For example:

```
/* skip blanks */  
while ( ( c = (char)System.in.read() ) == ' ' ) ;
```



assignment and boolean expression

Syntax of three loop statements

Loops containing a statement block:

```
while ( boolean_expression ) { [ statement; ] ... }
```

```
do { [ statement; ] ... } while ( boolean_expression );
```

```
for ( initializer ; boolean_expr ; increment ) {  
    [ statement; ] ... }
```

Each *statement* ends with a semi-colon,
but *no semi-colon after the block*.

```
/* skip blanks */  
do { c = (char) System.in.read();  
    } while ( c == ' ' ) ;
```

Repetition (loops) in Programs

Repetition (loop) is used a **lot** in computer programs.
Usually, repetition is used in computer programs like this:

Initialization Part

initialize data, open file,
ask user for input, ...



Loop Part

repeat operations in a loop



Conclusion Part

compute results, show
results or return a value

```
long sum = 0;  
Scanner input =  
new Scanner( System.in );
```



```
while ( input.hasNext( ) )  
    sum = sum +  
        input.nextInt( );
```



```
// results  
System.out.println(  
    "the sum is "+ sum );
```

Designing Repetition in Programs

You should think about these three activities when you write loops in your programs:

1. Initialization Part

what must be done first?

define and initialize variables, open files, print a prompt ...



2. Loop Part

repeat some actions until we are finished.

each time, test a condition to decide whether we are finished



3. Conclusion Part

process the results of the loop

print results or set attributes of an object or return results

while loop

English:

```
while more homework problems
  read the next problem
  solve it
```

Program:

```
while ( moreHomework( ) ) {
  getNextProblem( );
  solveProblem( );
}
// wasn't that easy?
```

while Loops for Reading Input

Three common approaches to reading input data:

- **exhaustive:**

 - read all the data until end of file or input stream

- **sentinel controlled:**

 - read data until you see a special value,
called a *sentinel*

- **counter controlled:**

 - read a count of expected number of data items,
then read exactly that many items

Q: which approach is used in the *Programming Skills Lab* ?

Reading input: exhaustive

How do you know when all the data has been read?

- Scanner: `hasNext()` method
- `BufferedReader`: returns `null`
- `InputStreamReader`: returns `-1`
- `DataInputStream`: throws `EOFException`

`DataInputStream` reads binary data, so it can't use `-1` or `null` to indicate EOF because those could be legitimate data values!

How To Find This Information?

How do you know what an input method does when EOF is found?
The details are stated in the Java API for each class that reads input.

Exhaustive Input Loop

Problem:

Read integers from the input and sum them.

English:

while there is more input
read the next number
add it to the sum

Program:

```
Scanner input = new Scanner( System.in );  
long sum = 0; // initialize the sum  
while ( input.hasNext( ) ) {  
    int data = input.nextInt( );  
    sum = sum + data;  
}
```

Exhaustive Input Loop (2)

Details for the careful programmer:

- (1) eliminate the temporary variable "n"
- (2) catch input errors

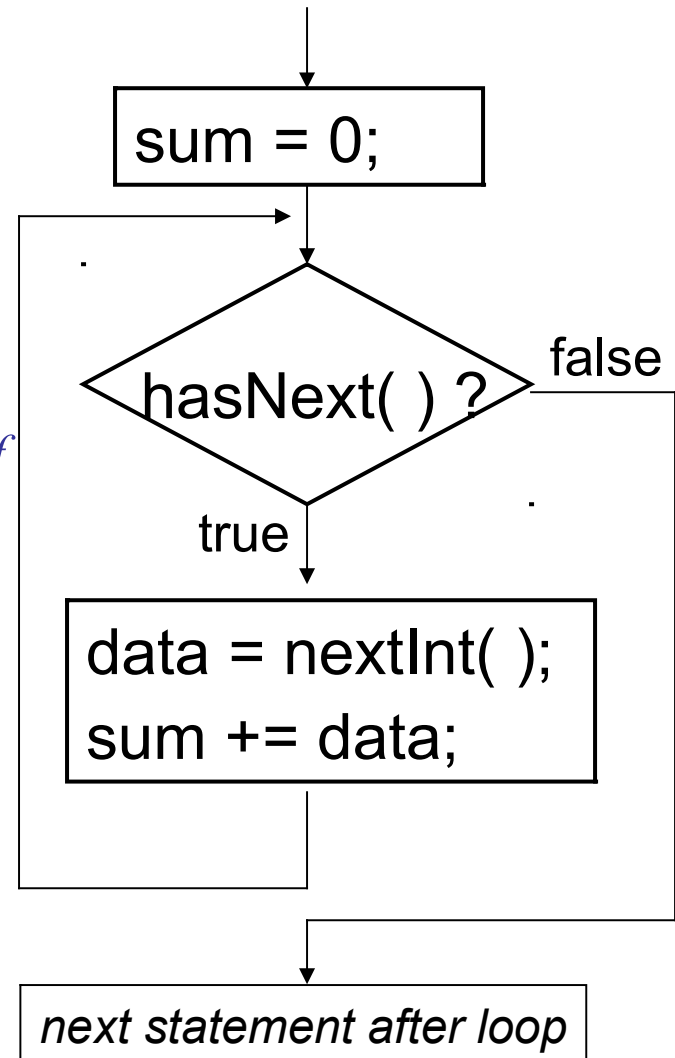
Program:

```
Scanner input = new Scanner( System.in );
long sum = 0; // initialize the sum
try {
    while ( input.hasNext( ) )
        sum += input.nextInt( );
} catch ( InputMismatchException e ) {
    // control comes here if an exception
    System.err.println("Read error: "+e);
}
```

Exhaustive Input Loop (3)

```
Scanner input =  
    new Scanner( System.in );  
long sum = 0; // initialize  
while ( input.hasNext( ) ) {  
    int data = input.nextInt();  
    sum = sum + data;  
}  
// Now sum = sum of all the  
// input values
```

*body of
loop*



Counter Controlled Input

Problem:

Read the number of expected values.

Read the expected amount of data and sum.

English:

read the counter value

while counter is greater than 0

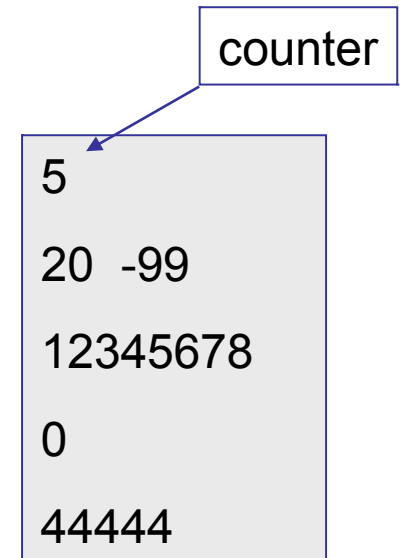
read the next number

add number to the sum

decrease the counter by 1

Program:

next slide



Counter Controlled Input (2)

Program:

```
Scanner input = new Scanner( System.in );
long sum = 0; // initialize the sum
int count = input.nextInt( );
while ( count > 0 ) {
    int data = input.nextInt( );
    sum = sum + data;
    count = count - 1;
}
// now sum = sum of the input values
```

Details for the careful programmer:

1. what could go wrong here?
2. how can you handle it?

Counter Controlled Input (3)

Details for the careful programmer:

1. there may be less data than we expect
2. there may be invalid counter or invalid data

Program with Error Checking:

```
Scanner input = new Scanner( System.in );
int count = 0, sum = 0; // initialize
try {
    count = input.nextInt( );
    while ( count > 0 ) {
        sum += input.getNextInt( );
        count--; // same as count = count -1
    }
} catch ( Exception e ) {
    System.err.println("Exception "+e +
        "\nafter reading "+ count +" values.");
}
```

Counter Controlled Input (4)

Details for the careful programmer:

instead of Exception you can use hasNextInt()

Program with Error Checking:

```
Scanner input = new Scanner( System.in );
int count = 0, sum = 0; // initialize
if ( input.hasNextInt( ) )
    count = input.nextInt( );
// test for more data AND counter
while ( count > 0 && input.hasNextInt() ) {
    sum += input.nextInt( );
    count--;
}
// now sum = sum of input values
System.out.println("The sum is "+sum);
```

Counter Controlled Input (4)

A Better Way:

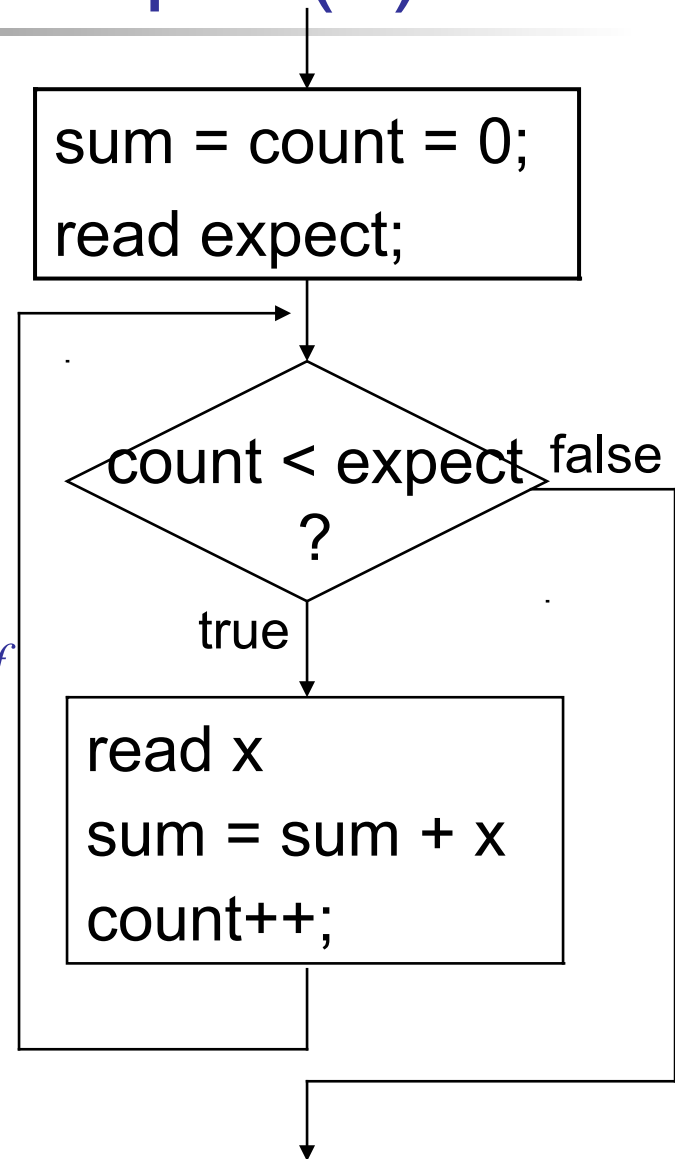
How can we save the number of items actually read? For example, to compute the average = sum/count

```
Scanner input = new Scanner( System.in );
long sum = 0;    // sum of the data
int count = 0;  // number of items read
int expect = input.nextInt( );
while ( count < expect ) {
    int data = input.nextInt( );
    sum = sum + data;
    count = count + 1;
}
// count = number of items actually read
if (count > 0 ) average = sum/count ;
```


Counter Controlled Input (5)

```
Scanner input =
    new Scanner( System.in );
long sum = 0; // sum of data
int count = 0; // number read
int expect= input.nextInt( );
while ( count < expect ) {
    int x = input.nextInt();
    sum = sum + x;
    count++;
}
// sum = total of the inputs
// count = number of values
if ( count > 0 )
    average = sum / count ;
```

} *body of loop*



Sentinel Controlled Input

Problem:

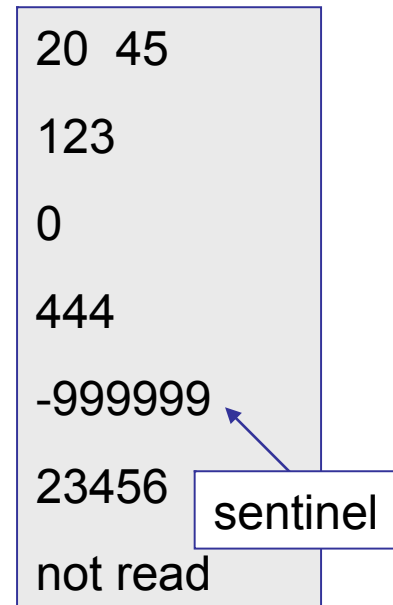
Read data from the input and sum values.
Stop when a special "sentinel" value is found

English:

set counter and sum to zero.
read first data value
while value not equal to sentinel
add value to the sum
increase the counter by 1
get the next value

Example:

next slide



Sentinel Controlled Input (2)

Example:

read the first value: $value = 20$

loop: if ($value == sentinel$) then stop (**false**)

add value to sum: $sum = 0 + 20$

read the next value: $value = 45$

loop: if ($value == sentinel$) then stop (**false**)

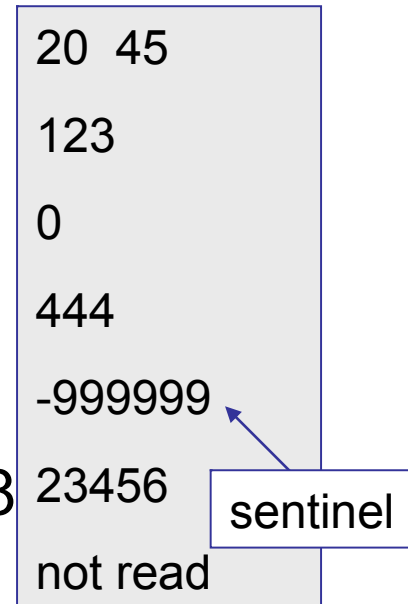
add value to sum: $sum = 20 + 45 = 65$

read the next value: $value = 123$

loop: if ($value == sentinel$) then stop (**false**)

add value to sum: $sum = 65 + 123 = 188$

read the next value: $value = 0$



Sentinel Controlled Input (3)

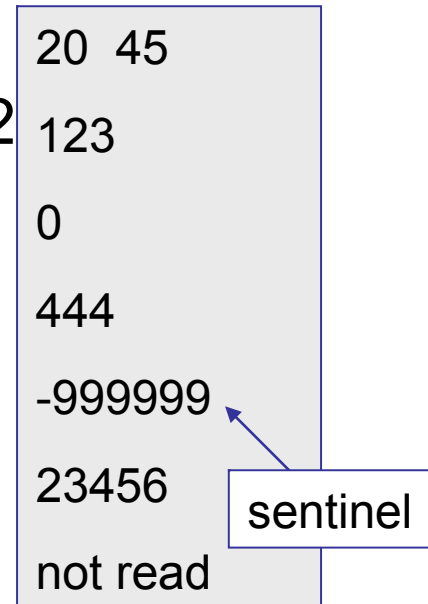
Example:

loop: if (value == sentinel) then stop (**false**)
add value to sum: $\text{sum} = 188 + 0 = 188$
read the next value: $\text{value} = 444$

loop: if (value == sentinel) then stop (**false**)
add value to sum: $\text{sum} = 188 + 44 = 232$
read the next value: $\text{value} = -999999$

loop: if (value == sentinel) then stop (**TRUE**)

finish: print the sum
 $\text{sum} = 232$



Sentinel Controlled Input (2)

Program:

```
final static int SENTINEL = -999999;
Scanner input = new Scanner( System.in );
long sum = 0; // initialize the sum
int count = 0; // count of data read
→ int data = input.nextInt( ); // first value
while ( data != SENTINEL ) {
    sum = sum + data;
    count++;
→ data = input.nextInt( ); // next value
}
// now sum = sum of the input values
```

Details: this code is slightly inefficient: duplicate input.

Sentinel Controlled Input (3)

Use a "do ... while" loop to eliminate duplication:

```
final static int SENTINEL = -999999;
Scanner input = new Scanner( System.in );
long sum = 0; // initialize the sum
int count = 0; // count of data read
do {
    data = input.nextInt( ); // next value
    if ( data != SENTINEL ) {
        sum = sum + data;
        count++;
    }
} while ( data != SENTINEL )
```

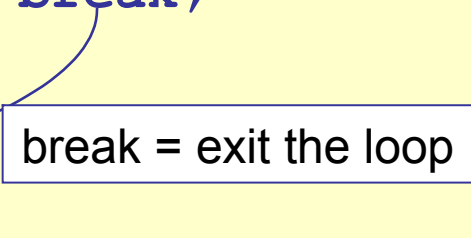
Details: still slightly inefficient: duplicate test for sentinel, no test for end of data.

Sentinel Controlled Input (4)

Use "break" to exit the loop:

```
final static int SENTINEL = -999999;
Scanner input = new Scanner( System.in );
long sum = 0; // initialize the sum
int count = 0; // count of data read

while ( input.hasNext( ) ) { // test
    data = input.nextInt( ); // next value
    if ( data == SENTINEL ) break;
    sum = sum + data;
    count++;
}
// now sum = sum of the input values
```



Details: could use "hasNextInt()" instead of "hasNext()".

Application of Sentinel Approach

Common Use:

Interactive applications: ask user a question and continue until he inputs a "quit" value.

```
Scanner input = new Scanner( System.in );  
do {  
    System.out.print("Choose option (0 to quit):");  
    int option = input.nextInt( );  
    if ( option != 0 ) processOption( option );  
while ( option != 0 );
```


Application of Sentinel Approach

Problems with Sentinel Approach for Data:

1. must be careful that real data cannot equal sentinel!
2. must still test for EOF -- what if user forgets sentinel or mistypes it?

```
BufferedReader br = new BufferedReader( ... );
String line;
while ( ( line = br.readLine() ) != null ) {
    if ( check_for_sentinel_value ) break;
    process input line;
}
```

Other while loop applications

The applications are endless! For example:

- ❑ searching a list **while** a desired value is not found
- ❑ repeat a calculation **until** the desired accuracy is achieved
- ❑ move on a chess board **while** there are vacant squares

```
// Find the Greatest Common Divisor of
// two integers. Example: gcd(30,72) = 6
public static int gcd(int a, int b) {
    a = Math.abs(a); // must be >= 0
    while ( b != 0 ) {
        int remainder = a % b;
        a = b;
        b = remainder;
    }
    return a;
}
```

Find the minimum and maximum value

Find the minimum and maximum of the input data.

```
// find the minimum and maximum of input data
float min, max, x;
// scanner object for reading the input
Scanner input = new Scanner( System.in );
x = input.nextFloat( ); // read first value
min = max = x;
while ( input.hasNext( ) ) {
    x = input.nextFloat();
    if ( x > max ) max = x;
    else if ( x < min ) min = x;
}
System.out.printf("min = %f   max = %f", min, max);
```

do ... while loop

English:

```
do
    make next move
while game is not finished
```

Program:

```
do {
    nextMove( )
} while ( ! gameOver( ) )
```

do ... while with Input

Problem:

Pick a number at random.

Ask the user to guess until he guesses right.

English:

choose a random integer 0 to some max value

explain the game to user

do

 prompt for user

 read guess

 evaluate guess and print the result

while guess is not correct

Guessing Game

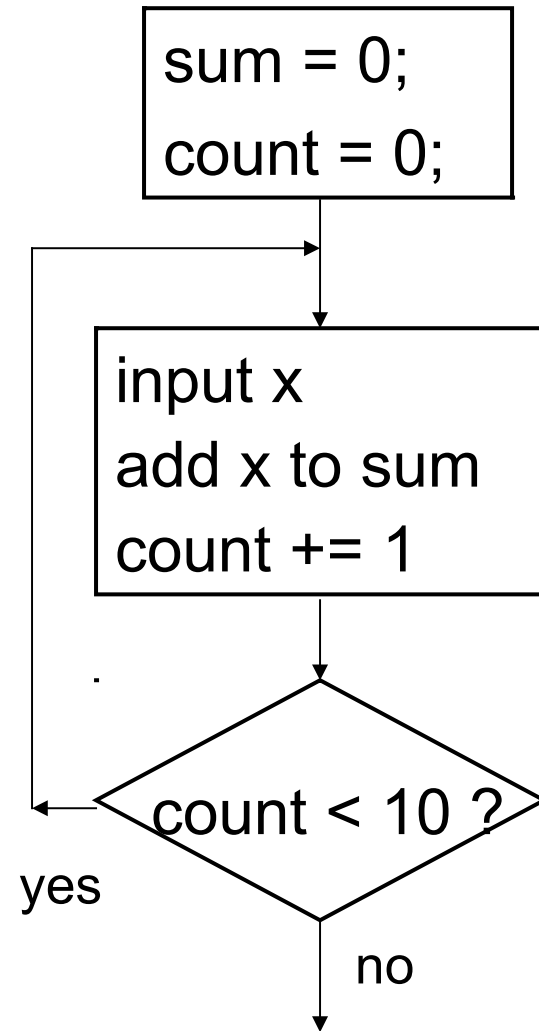
Program:

```
final int MAX = 20;
final int SECRET = (int) ( MAX * Math.random( ) );
System.out.printf(
    "Guess a number between 1 and %d\n", MAX);
int guess = 0;
do {
    System.out.print("Your guess: ");
    guess = input.nextInt( );
    if (guess == SECRET)
        System.out.print("right!");
    else System.out.println("Wrong. Try again.");
} while ( guess != SECRET );
```

Must end with semi-colon.

do *statement* while (*test_condition*);

```
float sum = 0, x;  
int count = 0;  
// read 10 number and sum  
do {  
    x = input.nextFloat( );  
    sum += x;  
    count++;  
} while ( count < 10 );  
float average =  
            sum / count;  
/* output the results */
```



`do statement while (test_condition);`

`do statement while (test)` is useful when:

- ❑ must execute some statements each time *before* the test condition can be performed.
- ❑ you want the loop *statement* to be executed at least once.

```
String reply;
do {
    PlayGame( );
    // ask user if wants to play again.
    reply = JOptionPane.showInputDialog(
        null, "Another game? " );
    if ( reply == null ) break;
} while ( ! reply.equalsIgnoreCase( "no" ) );
JOptionPane.showMessageDialog( null, "Goodbye" );
```


Gambling game... a random walk

Here is a game where you win or lose 1 Baht:

- Toss a coin.
 - If heads you win 1 Baht, if tails you lose 1 Baht.
- If you start with 10 Baht, can you get rich?

How to simulate a coin toss

- `Math.random()` returns a random number in [0 .. 1)
- each time, the result is different -- try it yourself!
- to simulate a coin toss (1/2 you win, 1/2 you lose) use:

```
if ( Math.random( ) > 0.50 ) /* you win 1 Baht */ ;  
else /* you lose 1 Baht */ ;
```

do ... while for Gambling Game

do

toss the coin.

if heads you win 1 Baht, else you lose 1 Baht

display amount of money you have

while (*your money* > 0)

```
int money = 10;    // you start with 10 Baht
double p = 0.50;  // probability of a "tail"
do {
    // toss the coin.  you win if "heads"
    if ( Math.random( ) > p ) money = money + 1;
    else money = money - 1;
    System.out.println("you have "+money);
} while ( money > 0 );
```

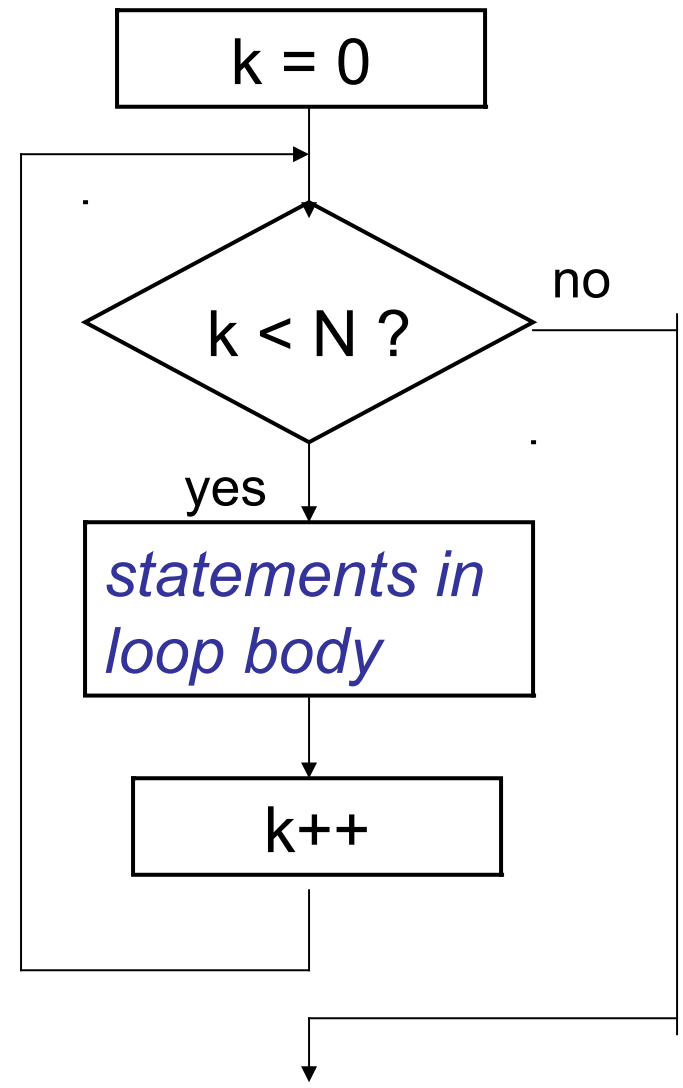
```
for ( initial ; test_condition ; increment )  
    statement ;
```

```
int k;  
for ( k = 0; k < N ; k++ ) {  
    // statements go here  
}
```

statement(s) to execute the first time only.

operation to perform at *bottom* of loop.

condition to test at *top of loop*



for loop examples

Here are some simple "for" loops

```
for(k = 2; k < 6; k++)  
    System.out.println("k = "+k);  
  
sum = 0;  
for(k = 5; k <= 20; k = k + 5 ) {  
    sum = sum + k;  
    System.out.printf(  
        "k = %2d sum = %d", k, sum );  
}
```

```
k = 2  
k = 3  
k = 4  
k = 5  
  
k = 5 sum = 5  
k = 10 sum = 15  
k = 15 sum = 30  
k = 20 sum = 50
```

for (*initial* ; *test* ; *increment*) *statement* ;

The "*test*" condition is tested at the start of each iteration.

You can verify this using this experiment:

```
int k, N;  
System.out.print("Please input N:");  
N = scanner.nextInt( );  
for ( k = 0; k < N ; k++ ) {  
    System.out.println("now k = "+k);  
}  
System.out.println("done. k = "+k);
```

Please input N: **1**

now k = 0

done. k = 1

(run program again...)

Please input N: **0**

done. k = 0

body of loop is not executed

for (*initial* ; *test* ; *increment*) *statement* ;

The *initial*, *test*, or *increment* may be omitted.
But you must still include the semi-colons.

```
int k;  
int n = 10;  
System.out.print("What value for start of loop?");  
k = scanner.nextInt( );  
  
for ( ; k < N ; k++ ) {  
    System.out.printf("k=%d\n", k);  
}  
System.out.println("done");
```

"for" loop for processing array

A common use of "for" is to process all elements of an array.

```
float [] a;  
// read the data from somewhere  
a = readArrayData( ); // returns an array  
  
// sum the values and find the maximum value  
float sum = 0;  
float max = a[0];  
for ( int k=0 ; k < a.length; k++ ) {  
    sum += a[k];  
    if ( a[k] > max ) max = a[k];  
}
```

common "for" loop usage

break - *exit a loop*

The break statement causes execution to leave the **innermost** surrounding for, while, or do...while loop and continue at the next statement after loop.

```
// average of numbers, stop if sum > 1E6
double sum, x,;
int count = 0;
sum = 0.0;
while ( sum < 1.0E6 ) {
    if ( ! scanner.hasNextDouble( ) ) break;
    x = scanner.nextDouble( );
    sum += x;
    count++;
}
```


break with label - *exit any loop*

In the case of nested loops, to break out of the outer loop you must use a *label*. To label a statement, do this:

```
label: statement;
```

```
// a double loop over an array
final int ROWS = 10, COLS = 20;
double [][] x = new double[ROWS][COLS];
// label the outer loop
ROWLOOP: for ( int row=0; row<ROWS; row++ ) {
    for ( int col=0; col<COLS; col++ ) {
        if ( ! scanner.hasNextDouble( ) )
            break ROWLOOP;
        x[row][col] = scanner.nextDouble( );
    } // end of "col" loop
} // end of "row" loop
```

continue - *go to next iteration of loop*

The continue statement causes execution to skip the remaining statements in a for, while, or do...while loop and perform the test/increment for next iteration.

```
int n;  
// print numbers 1 .. 20  
// except multiples of 4  
for ( n = 0; n < 20; n++ ) {  
    if ( n % 4 == 0 ) continue;  
    System.out.println(n);  
}
```

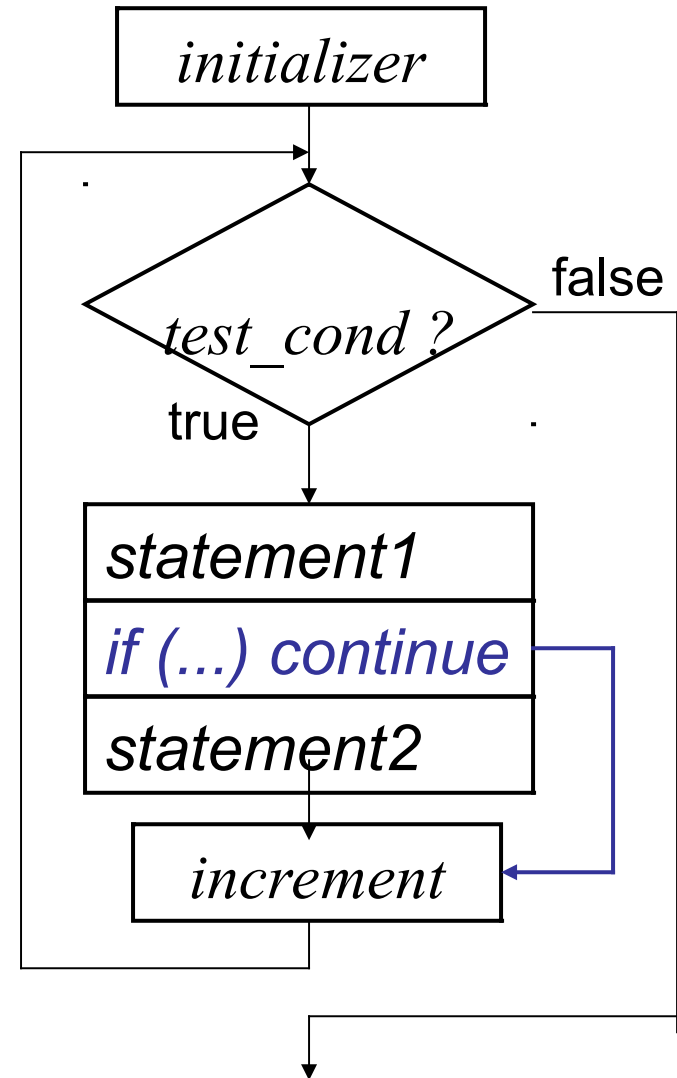
1
2
3
5
6
7
9
10
11
13
14

15
17
18
19

continue - go to next iteration of loop

In a while() or do...while() loop, continue branches to the test condition. In for(), it branches to the increment part.

```
for ( initializer ; test_cond ;  
    increment_operation )  
{  
    body of loop;  
    if ( something ) continue;  
    more statements;  
}
```



Weird "for" loops

Sometimes you will see "for" loops used in unusual ways.
Try to avoid writing loops like this.

for (*initial* ; ; *increment*) *statement* ;

Sometimes the test condition is performed inside the “for” loop, so you can omit it in the for statement.

```
int count;
// sum numbers until a zero
// is found
for ( count = 0 ; ; count++ ) {
    x = input.nextInt( );
    if ( x == 0 ) break;
    sum += x;    count++;
}
System.out.println("Sum is: "+
    sum);
```

break causes the flow of execution to leave a for, while, or do...while loop and continue at first statement after the loop.

for (*initial* ; *test* ;) *statement* ;

You can omit the *increment* part of a for statement, but in that case using a while () ... loop is clearer.

```
float min, max, x;  
// find the minimum and maximum  
x = input.nextFloat( );  
for ( min = x, max = x ;  
      input.hasNext();  
      ) {  
    x = input.nextFloat;  
    if ( x > max ) max = x;  
    else if ( x < min ) min = x;  
}
```

initialize both min
and max

true while there
is more input to
read

no increment
operation

for (*initial* ; *test* ;) is same as while

If you omit the *increment* part of a for statement, then using a while loop is clearer.

```
// find the minimum and maximum of input data
float min, max, x;
// scanner object for reading the input
Scanner input = new Scanner( System.in );
x = input.nextFloat( ); // read first value
min = max = x;
while ( input.hasNext( ) ) {
    x = input.nextFloat();
    if ( x > max ) max = x;
    else if ( x < min ) min = x;
}
System.out.printf("min = %f  max = %f", min, max);
```

Any statement can be used in "for(...)"

This works but is hard to read:

```
double sum, x;
int count = 0;
Scanner in = new Scanner(System.in);

// read numbers until a negative value is found
for ( System.out.println("Input numbers: ") ;
      in.hasNext() && (x=in.nextInt())>=0 ;
      sum += x ) count++;

System.out.println("The Sum is: "+ sum);
```