COP 3330: Object-Oriented Programming Summer 2007

Introduction to Object-Oriented Programming Part 3

Instructor : Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 823-2790
http://www.cs.ucf.edu/courses/cop3330/sum2007

School of Electrical Engineering and Computer Science
University of Central Florida

Simple Boolean Expressions

- There are only two possible values for a Boolean expression:
  - `true` `false`
- A simple boolean expression (not all of them) can have the following form:
  - *expression1 relational-operator expression2*
- Relational Operators:

`<`	less than
`<=`	less than or equal to
`>`	greater than
`>=`	greater than or equal to
`==`	equal to (do not confuse with =)
`!=`	not equal to

Simple Boolean Expressions -- Examples

- If x is 10 and y is 5 (int x = 10, y=5)

Boolean Expressions		*Results*
x > 1	➔	true
x < 5	➔	false
x >= 10	➔	true
y < x	➔	true
y != 5	➔	false
x == y	➔	false
y <= x	➔	true

Boolean Expressions

- Boolean Operators

&&	logical AND
||	logical OR
!	negation (NOT)
^	exclusive or (XOR)

- A Boolean expression is:
  - a Boolean literal or a Boolean variable
  - a simple Boolean expression with a relational operator
    - *operand1 relational-operator operand2* where operands are arithmetic operations (most of the time)
  - a Boolean expression with a boolean operator
    - *operand1 Boolean-operator operand2*
      - where operands are Boolean expressions
    - *Boolean-operator operand*
      - where operand is a Boolean expression

Boolean Expressions -- Examples

- If x is 10 and y is 5 (int x = 10, y=5)

Boolean Expressions		*Results*
(x>1) && (y<5)	➔	false
(x>1) || (y<5)	➔	true
!(y<5)	➔	true
(x<1) ^ (y>1)	➔	true

Truth Tables of Boolean Operators

E1	E2	E1 && E2	E1 || E2	E1 ^ E2
false	false	false	false	false
false	true	false	true	true
true	false	false	true	true
true	true	true	true	false

E	!E
false	true
true	false

Operator Precedence and Associativity

Precedence Level	Operator	Operation	Associates
1	++ --	postfix increment, postfix decrement	L to R
2	++ -- + - ~ !	pre-increment, post-increment, unary minus and plus bitwise complement logical not	R to L
3	* / %	multiplication, division, remainder	L to R
4	+ -	addition, subtraction	L to R
5	<< >> >>>	left shift, right shift with sign, left shift with zero	L to R
6	< <= > >=	less than, less than equal, greater than, greater than equal	L to R
7	== !=	equal, not equal	L to R
8	&	bitwise and	L to R
9	^	xor	L to R
10	|	bitwise or	L to R
11	&&	logical and	L to R
12	||	logical or	L to R
13	?:	conditional operator	R to L
14	= +=	assignment operators	R to L

Operator Precedence -- Example

(x>1) || (y<2) && (z==3) && is evaluated first

(x>1) && (y<2) && (z==3) left to right

(x>1) || (y<2) || (z==3) left to right

Boolean Assignment:

- we may declare Boolean variables and save Boolean values in those variables.

```
boolean flag;
flag = (x>y);
```


Short Circuit Evaluation

- If the value of the first operand of an && operator is `false`, the second operand is not evaluated.
- If the value of the first operand of an || operator is `true`, the second operand is not evaluated.

(x>y) && (x>z) if (x>y) is `false` the value of (x>z) is not evaluated

(x>y) || (x>z) if (x>y) is `true` the value of (x>z) is not evaluated

- We have to be careful when using operations with side effects.

  int x=2;

  (x > 5) && (x++ <10) ➔ false, x is not incremented

  (x < 5) && (++x < 10) ➔ true, x is incremented

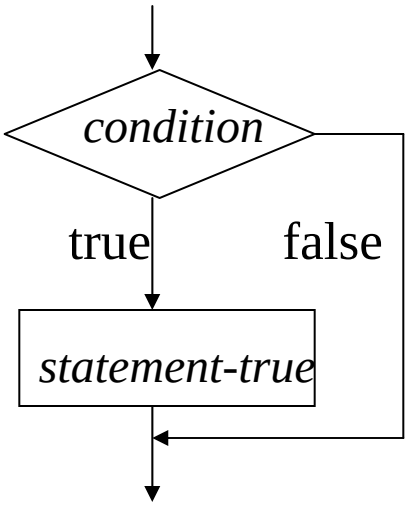
  (x++ < 10) && (x > 5) ➔ false, x is incremented

if Statement

if-then structure:

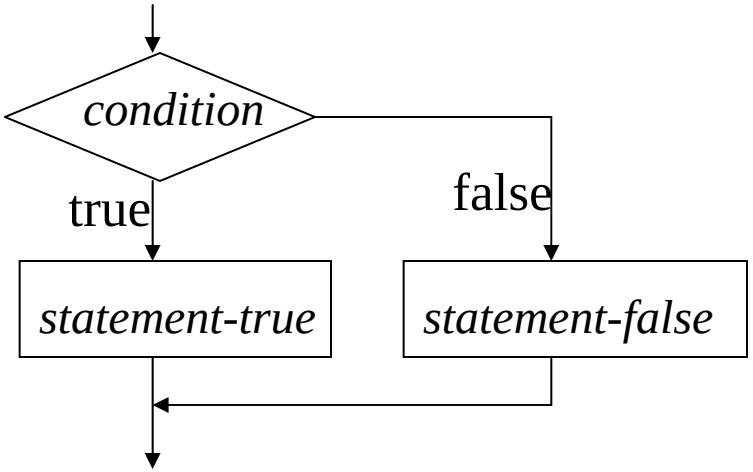
```
if ( condition )
    statement-true
```

where *condition* is a Boolean expression, and *statement-true* is any statement;



if-then-else structure:

```
if ( condition )
    statement-true
else
    statement-false
```



if-statement Examples

```
if (y != 0)
   x = x / y;
else
   System.out.println("y is zero");
```

```
if (x >= 0)
   System.out.println("positive");
else
   System.out.println("negative");
```

```
if (x < 0)
   sign = -1;
else
   sign = 1;
absolutex = sign * x;
```

```
if((x>1) && (x<10))
   x = x + 1;
```

Compound Statements

- A set of statement contained within a pair of braces { and }, is called a **compound statement**.
- The body of a method is also a compound statement.
- A compound statement can be used anywhere in a program that a single statement can be used.
- After a compound statement, we should not put a semicolon. If we put one, this means that we have inserted an empty statement.

Compound Statement -- Examples

```
if (discountRate != 0) {
   discount = price * discountRate;
   price = price – discount;
}
```

```
if (x>y) {
   temp = x ;
   x = y;
   y = temp;
}
```

```
if (x != y) {
   System.out.println(“x and y have different values”);
   System.out.println(“x: “+x+” y: “+y);
}
else
   System.out.println(“x and y have same value: “ + x);
```


Nested if-statements

- If-statements can also be used to implement decisions involving more than two alternatives.
- A nested if-statement is an if-statement whose true-statement and/or false-statement are also if-statements.

```
if (x < 10)
   System.out.println(“x has one digit”);
else if (x < 100)
   System.out.println(“x has two digits”);
else
   System.out.println(“x has more than two digits”);
```


Nested if-statement Example

```
if (grade >= 90)
  System.out.println("A");
else if (grade >= 80)
  System.out.println("B");
else if (grade >= 70)
  System.out.println("C");
else if (grade >= 60)
  System.out.println("D");
else
  System.out.println("F");
```


Dangling Else Problem

- Java compiler always associates an else-statement with the previous if-statement (the closest one). This can only be changed by using a compound statement.

THE FOLLOWING EXAMPLE IS INCORRECT!! {try x = 7, y = 3}

```
if (x > 5)
   if (y > 5)
         System.out.println("x and y are greater than 5");
   else
        System.out.println("x is less than or equal to 5");
```

THIS IS THE CORRECT WAY TO ACHIEVE THE PROPER RESULT

```
if (x > 5) {
   if (y > 5)
         System.out.println("x and y are greater than 5");
   }
   else
         System.out.println("x is less than or equal to 5");
```


An Example Program with If-Statements

```
// Finding the minimum of given three integers – Application Version
import java.io.*;

public class MinTest
{  public static void main(String args[]) throws IOException
   {    int minimum, num1, num2, num3;
        BufferedReader stdin = new BufferedReader(new
                                        InputStreamReader(System.in));
        // read three integers
        System.out.println("Enter three integers : ");
        num1 = Integer.parseInt(stdin.readLine().trim());
        num2 = Integer.parseInt(stdin.readLine().trim());
        num3 = Integer.parseInt(stdin.readLine().trim());
        // find the minimum of three numbers
        if (num1 < num2)
           minimum = num1;
        else
           minimum = num2;
        if (num3 < minimum)
           minimum = num3;
        // display the result
        System.out.println("The minimum number is: " + minimum);
} }
```


Switch Statement

```
switch ( expression ) {
   case value1 :
        statement-list1
   case value2 :
        statement-list2
        .
        .
   case value4 :
        statement-listn
   default :
        statement-listd
}
```

each *statement-list* is a sequence of statements, and normally it ends with a `break` statement.

we may use a switch statement to select one alternative from many alternatives.

Switch Statement (cont.)

```
int x;
...
switch (x) {
   case 1:
       System.out.println(“x is 1”);
       break;
   case 2:
       System.out.println(“x is 2”);
       break;
   default :
       System.out.println(“x is not 1 or 2”);
}
```


Switch Statement (cont.)

- A switch statement can be implemented as a nested if-statement.

```
if (x==1)
  System.out.println(“x is 1”);
else if(x==2)
  System.out.println(“x is 2”);
else
  System.out.println(“x is not 1 or 2”);
```


Switch Statement (cont.)

- Break statements are necessary in case parts.

```
switch (x) {
  case 1:
    System.out.println(“x is 1”);
  case 2:
    System.out.println(“x is 2”);
    break;
  default :
    System.out.println(“x is not 1 or 2”)
}

if (x==1)
   { System.out.println(“x is 1”); System.out.println(“x is 2”) }
else if(x==2)
   System.out.println(“x is 2”);
else
   System.out.println(“x is not 1 or 2”);
```


while Statement

- Repetition of a group of statements is called as **loop**.
- There are more than one statement to handle loops in Java.
- The most general one of these loop statements is **while**.

`while` (*boolean-expression*)

statement

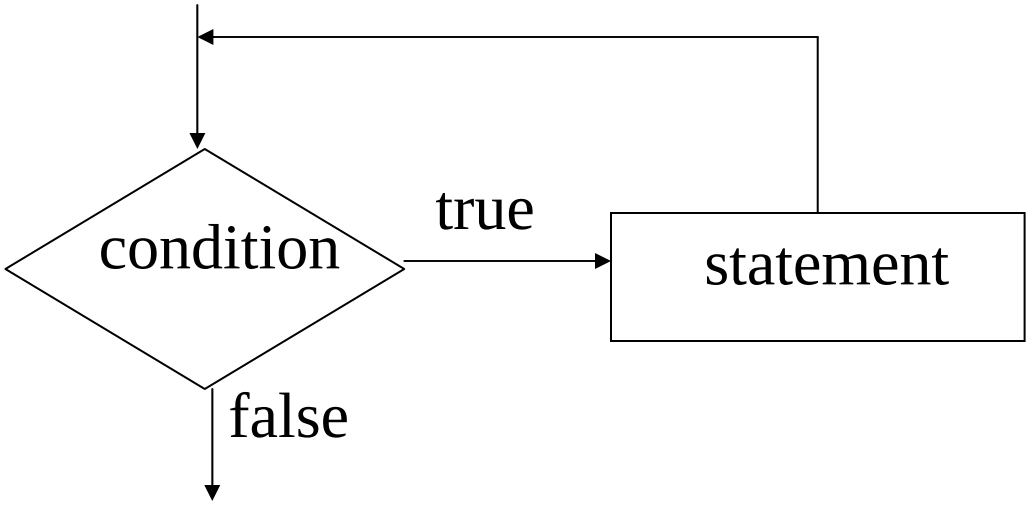
where statement can be any statement including a compound statement, an if-statement, another loop statement, ...

- Statement is repeated as long as the condition (boolean-expression) is true. When the condition gets false, the statement is not executed anymore.
- If the condition never gets false ➔ infinite loop
- If the condition gets immediately false ➔ the loop will be repeated zero times.

while Statement – Flow Diagram

`while` (*condition*)

statement



while Statement -- Example

```
int countName = 0;
String aname;
while (countName < 10) {
   System.out.print(“Enter a name > “);
   System.out.flush();
   aname = stdin.readLine();
   System.out.println(“Hi “ + aname);
   countName = countName + 1;
}
System.out.println(“All names are processed.”);
```

- This loop will be repeated for 10 times (countName: 0,1,...,9)

Sentinel-Controlled Loops

- What happens if we don't know how many times a loop will run.
- Ex: a loop which reads the scores of the students in an exam and find the average of the scores; and we we don't know the number of students. How are we going to stop the loops?

➔ use a **sentinel-value**

- We choose a sentinel-value which can not be a score (e.g. –1)
- We read the scores until this sentinel-value has been entered. When this sentinel-value has been read, we stop the loop.

Sentinel-Controlled Loops – Example1

```
int sum = 0;
int numOfStudents = 0;
int ascore;
double avg;
System.out.print("Enter a score (-1 to stop) >");
System.out.flush();
ascore = Integer.parseInt(stdin.readLine().trim());
while (ascore != -1) {
   sum = sum + ascore;
   numOfStudents = numOfStudents + 1;
   System.out.print("Enter a score (-1 to stop) >");
   System.out.flush();
   ascore = Integer.parseInt(stdin.readLine().trim());
}
avg = (double) sum / numOfStudents;
System.out.println("The number of students : " +
   numOfStudents);
System.out.println("Average Score : " + avg);
```

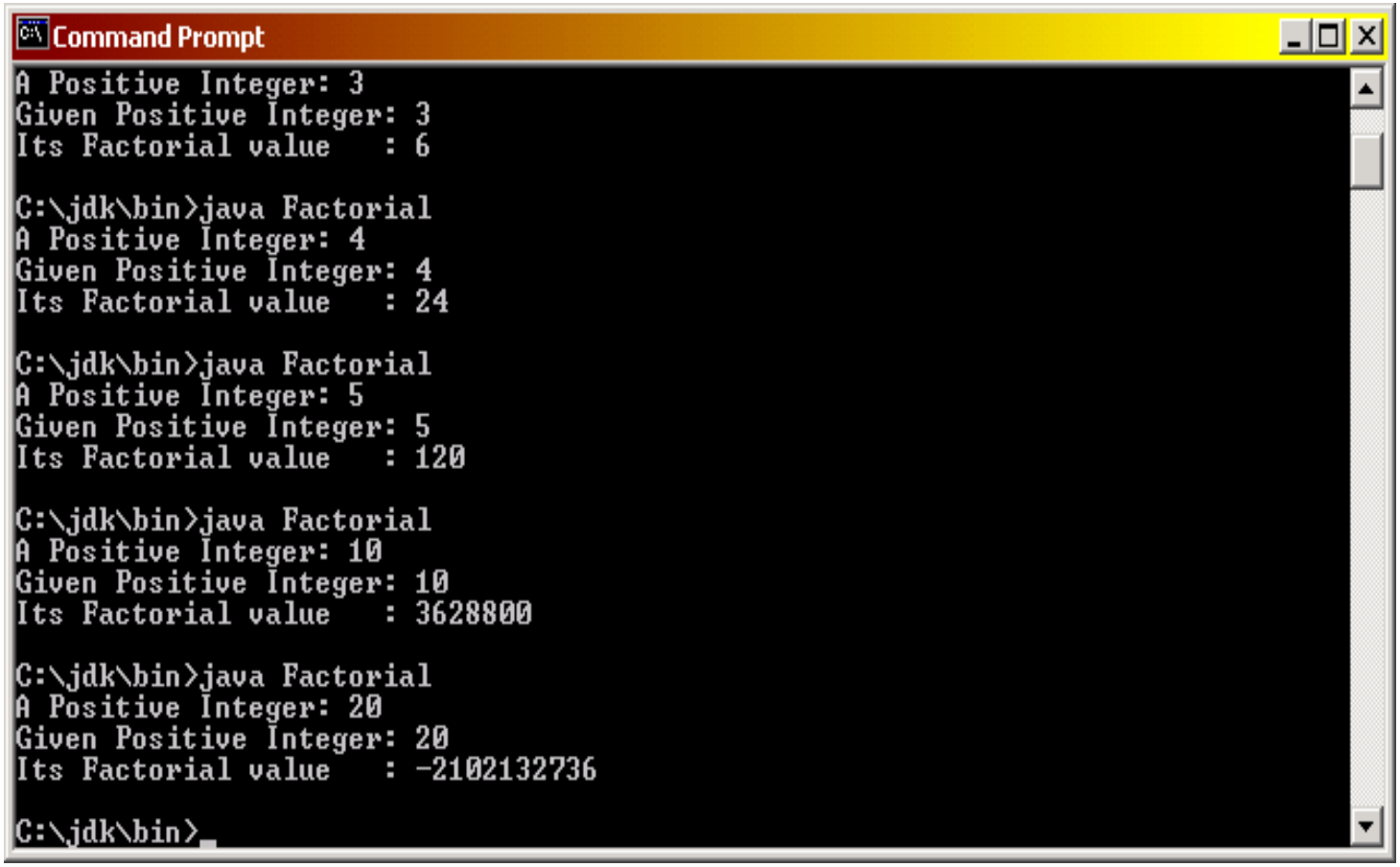

Finding The Smallest Number -Example

```
int smallest = 2147483647;   // maximum positive int value
int num
System.out.print("Enter a positive integer(negative to
   stop) >");
System.out.flush();
num = Integer.parseInt(stdin.readLine().trim());
while (num > -1) {           // as long as a positive integer is entered
   if (num < smallest)
       smallest = num;
   System.out.print("Enter a positive integer(negative to
   stop) >");
   System.out.flush();
   num = Integer.parseInt(stdin.readLine().trim());
}
System.out.println("The smallest positive integer entered:
   " + smallest);
```


Example Program

```
// This program finds the factorial value of the given positive integer
import java.io.*;
public class Factorial {
  public static void main(String args[])throws IOException {
    int num, factVal, counter;
    // Create BufferedReader object
    BufferedReader stdin =
    new BufferedReader (new InputStreamReader (System.in));
    // Read the positive integer
    System.out.print("A Positive Integer: ");
    System.out.flush();
    num = Integer.parseInt(stdin.readLine().trim());
    // Find its factorial value
    counter = 2;
    factVal = 1;
    while (counter <= num) {
      factVal = factVal * counter;
      counter = counter + 1;
    }
    // Write the result
    System.out.println("Given Positive Integer: " + num);
    System.out.println("Its Factorial value   : " + factVal);
  } }
```


Output of Factorial Program



Example Program – Plus/Minus Counter

```
// This program finds the number of positive and negative numbers which are
// entered from the keyboard. Each integer is entered from a separate line,
// and the last number 0 (which is not treated as positive or negative number).
// At the same time, this program finds the position of the first positive number
// and the position of its second occurrence.
// This program prints the number of positive and negative number. If the numbers
// contains at least one positive number, the position of the first positive
// integer and the position of its second occurrence (if the first positive number
// does not occur second times, we print 0 for second occurrence) are printed.

import java.io.*;
public class PlusMinusCounter {
 static final int SENTINEL = 0;    // The last number
 public static void main(String args[])throws IOException {
   int anInt,              // the number entered
       numOfPos,           // number of positive numbers
       numOfNeg,           // number of negative numbers
       locOfFirstPos,      // position of first positive number
       secLocOfFirstPos, // pos. of second occurrence of 1st positive number
       firstPos,           // the first positive number
       loc;                // current location
```


Plus/Minus Counter Program (cont.)

```
// Create BufferedReader object
BufferedReader stdin =
    new BufferedReader (new InputStreamReader (System.in));
// Initialize the variables
loc = 0;  numOfPos = 0;  numOfNeg = 0;
firstPos = 0; locOfFirstPos = 0;  secLocOfFirstPos = 0;
// Read the first number
System.out.print("Enter an integer (0 to stop) > ");
System.out.flush();
anInt = Integer.parseInt(stdin.readLine().trim());
// Find the number of positive and negative numbers.
// At the same time, find the position of the first positive number
// and the position of its second occurrence.
while (anInt != SENTINEL) {
         loc = loc + 1;
         if (anInt<0)
            numOfNeg = numOfNeg + 1;    // negative number
         else {  // positive number
            numOfPos = numOfPos + 1;
```

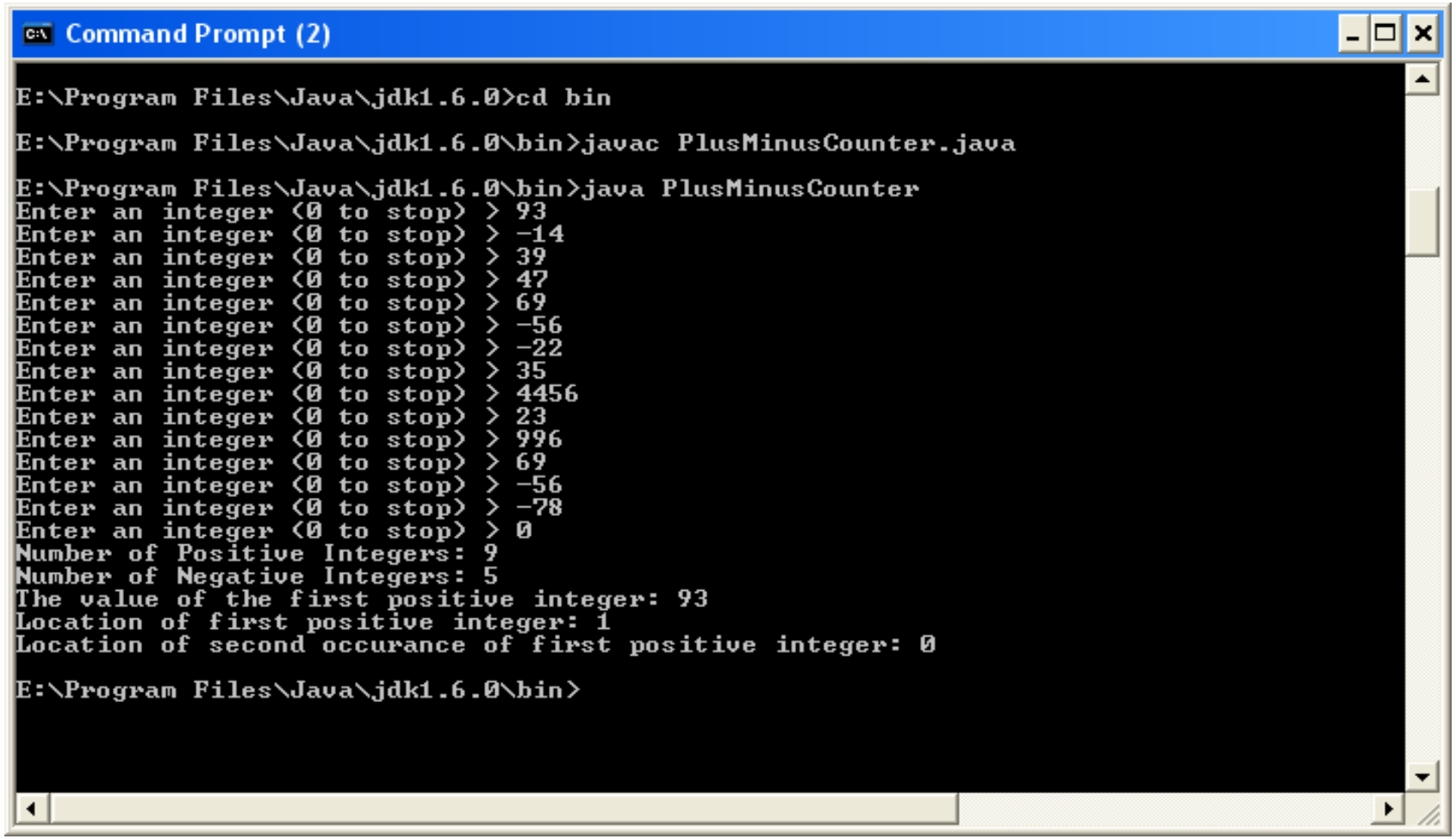

Plus/Minus Program (cont.)

```
if (locOfFirstPos==0) {     // first positive number
               firstPos = anInt;
               locOfFirstPos = loc;
            }
            else if((secLocOfFirstPos==0) && (firstPos==anInt))
               // second occurrence of the first positive number
              secLocOfFirstPos = loc;
         }
         // read the next number
         System.out.print("Enter an integer (0 to stop) > ");
         System.out.flush();
         anInt = Integer.parseInt(stdin.readLine().trim());
      }
// Display the results
System.out.println("Number of Positive Integers: " + numOfPos);
System.out.println("Number of Negative Integers: " + numOfNeg);
if (locOfFirstPos!=0) {
   System.out.println("The value of the first positive integer: "
      + firstPos);
```


Plus/Minus Program – (cont.)

```
System.out.println("Location of first positive integer: "
      + locOfFirstPos);
   System.out.println(
      "Location of second occurance of first positive integer:
   "
      + secLocOfFirstPos);
}
}
}
```


Output of Plus/Minus Counter Example



for Statement

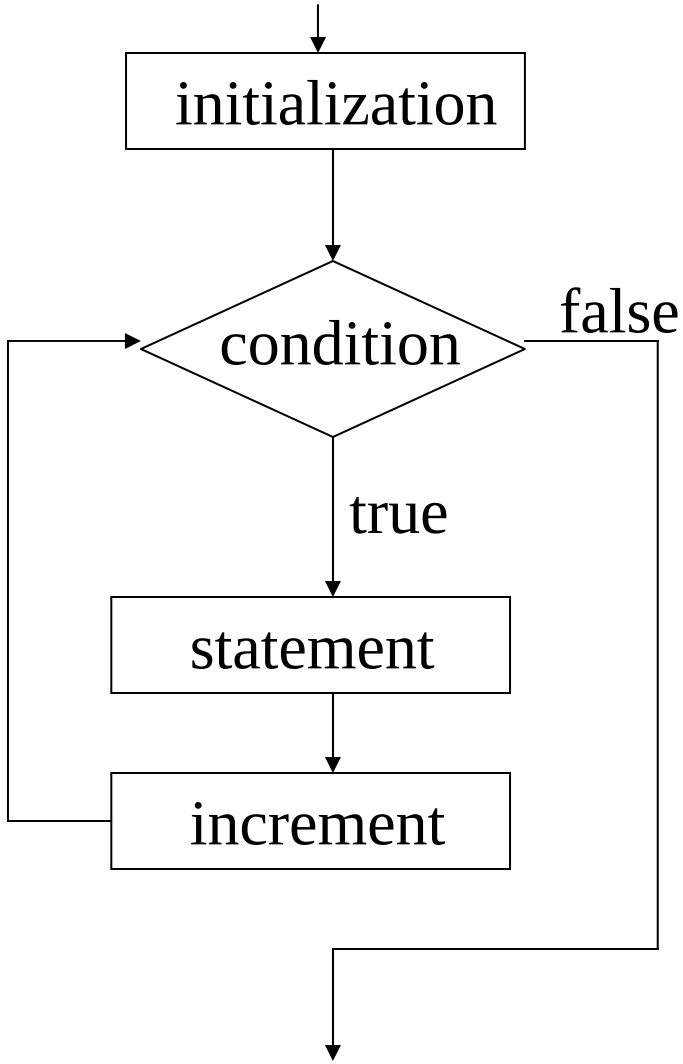
- Another loop statement in Java is **for-statement**.
- for-statement is more suitable for counter-controlled loops.

```
for ( initialization ; condition ; increment )
      statement
```

which is equivalent to

```
initialization ;
while (condition ){
      statement ;
      increment ;
}
```


for Statement – Flow Diagram



for statement -- Examples

```
int i;
int sum = 0;
for (i=1; i<=20; i++)
   sum = sum + i;
```

```
int i;
int sum = 0;
      i = 1;
      while (i<=20) {
        sum = sum + i;
        i++;
      }
```

```
int i;
int sum = 0;
for (i=100; i>=1; i--)
   sum = sum + i;
```

```
      int i;
int sum = 0;
      i = 100;
      while (i>=1) {
        sum = sum + i;
        i++;
      }
```

for statement -- More Examples

```
int i, j;
int count=0;
for (i=1, j=10; i<j; i++, j--)          ➔ count is 5
   count++;
```

```
int i, j;
int count=0;
i=1;
j=10;
for (; i<j; ) {
   count++;
  i++;
  j--;
}
```


do-while statement

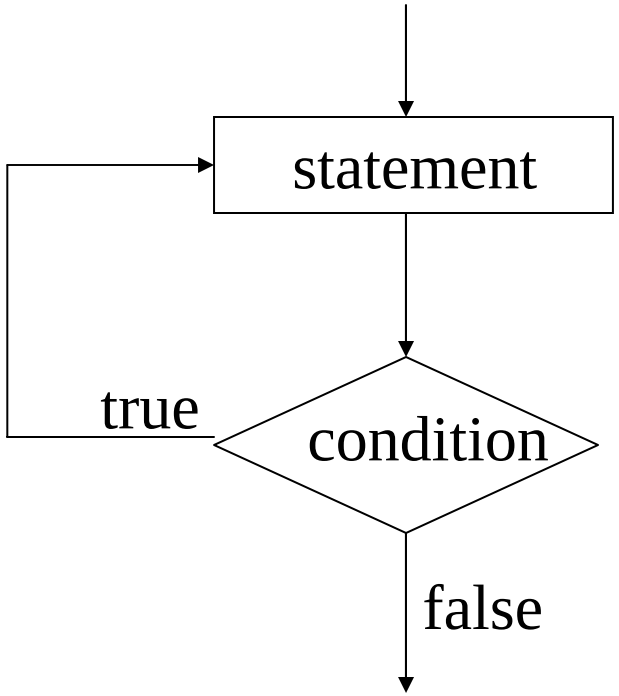
- The third loop statement in Java is the do-while statement.

```
do
    statement
while ( condition ) ;
```

which is equivalent to

```
statement
while (condition )
     statement
```


do-while statement – Flow Diagram



do-while statement -- Example

```
int i=1;
do {
   System.out.println(i);
   i++;
} while (i<10);
```

```
String s;
do {
   System.out.print("Enter a word > ");
   System.out.flush();
   s = stdin.readLine();
   System.out.println(s);
} while (! s.equals("quit"));
```


Nested Loops

- If the body of a loop contains another loop, this is called as a **nested loop**.

```
int sum=0;  int i,j;
for (i=1; i<=5; i++)
  for (j=1; j<=6; j++)
    sum=sum+1;                    ➔ sum is 5*6=30

int sum=0;  int i,j;
for (i=1; i<=5; i++)
  for (j=1; j<=i; j++)
    sum=sum+1;                    ➔ sum is 1+2+3+4+5=15
```


Nested Loops – Another Example

- a right angled triangle (n lines, ith line contains i stars)

```
for (i=1; i<=n; i++) // for each line
{   for (j=1; j<=i; j++)
            System.out.print("*");
            System.out.println("");
}
```


Nested Loops – Yet Another Example

- a triangle shape (1st line contains 1 star, 2nd line contains 3 star,..., nth line contains 2n-1 stars)

```
   *          for (i=1; i<=n; i++) { // for each line
  ***                   for (j=1; j<=(n-i); j++)
 *****                    System.out.print(" ");
   .                  for (j=1; j<=(2*i-1); j++)
   .                    System.out.print("*");
***.....***           System.out.println("");
                     }
```


Nested Loops

- Nesting can be more than one level

```
int sum=0;
for (i=1; i<=5; i++)
  for (j=1; j<=5; j++)
    for (k=1; k<=5; k++)
      sum=sum+1;
```

➔ sum is 125

An Example Applet Using Loops

```
// This applet draw a ladder like shape
// The number of the steps is read from a TextField
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class Ladder extends Applet implements
   ActionListener{
    private final int MAXSTEPS = 30;
    private int  numOfSteps = 0;
    private TextField input;

    // Create components and put them into the applet
    public void init() {
      add(new Label("Number of Steps: "));
      input = new TextField(10);
      add(input);
      input.addActionListener(this);
    }
```


Applet with Loops (cont.)

```
// Draw the ladder like shape
public void paint(Graphics g) {
   int i;
   int x=20, y=20;
   for (i=0; i<numOfSteps; i=i+1) {
       g.drawLine(x,y,x+20,y);
       g.drawLine(x+20,y,x+20,y+20);
       x=x+20;
       y=y+20;
   }
 }

 // Read the number of steps and activate paint method
 public void actionPerformed(ActionEvent e) {
    numOfSteps =
Integer.parseInt(e.getActionCommand().trim());
    if (numOfSteps>MAXSTEPS)
         numOfSteps = MAXSTEPS;
    else if (numOfSteps<0)
         numOfSteps = 0;
    repaint();
 } }
```


The Applet -- Output