More about

Flow of Control

Topics

- Debugging
- Logical Operators (Chapter 5)
- Comparing Data (Chapter 5)
- The conditional operator
- The switch Statement
- The for loop

Debugging

- Your program doesn't work, and you don't see what's wrong by looking at the code.
- What do you do?
  - Add println statements to show what's happening as the program runs.

Warning: This example has errors

```
//*************************************************************
// Name: xxxxx                File: Extremes.java
//
// Finds maximum, minimium, and average of up to 10 integers
// entered from the keyboard.
//*************************************************************

import java.text.DecimalFormat;
import java.util.Scanner;

public class Extremes
{
    public static void main(String[] args)
    {
        int n, min, max, sum=0, count=0;
        Scanner scanner=new Scanner(System.in);

        System.out.print("Enter number: ");  // Get first input
        n = scanner.nextInt();
        max = n;
        min = n;
```

Warning: This example has errors

```
while (n != 0)
{
    count++;
    sum += n;
    System.out.print("Enter number: "); // Get additional inputs
    n = scanner.nextInt();
    if (n > max)
    {
        max = n;
    }
    if (n < min)
    {
        min = n;
    }
}
```

Warning: This example has errors

```
        if (count == 0)
        {
            System.out.println("No numbers entered");
        }
        else
        {
            System.out.println("The max value is: " + max);
            System.out.println("The minimum value is: " + min);
            System.out.println("The sum is: " + sum);
            double avg = (double) sum / count;

            DecimalFormat fmt= new DecimalFormat("0.####");
            System.out.println("The average of the values " +
                                fmt.format(avg));
        }
    }
}
```

Test Run

Command Prompt

```
C:\test>
C:\test>
C:\test>
C:\test>javac Extremes.java

C:\test>java Extremes
Enter number: 1
Enter number: 2
Enter number: 3
Enter number: 0
The max value is: 3
The minimum value is: 0
The sum is: 6
The average of the values is 2

C:\test>_
```

Look at the code

- Is min initialized to 0?
- No, both min and max are initialized to the value of the first input.
- Add println statements to tell what is happening as the program runs.

Add println statements

- At end of the while loop

```
System.out.println(
    "count = " + count + "  min = " + min);
```

Save, compile, and run

Program Running

Command Prompt

```
C:\test>
C:\test>
C:\test>javac Extremes.java

C:\test>java Extremes
Enter number: 1
Enter number: 2
count = 1  min = 1
Enter number: 3
count = 2  min = 1
Enter number: 0
count = 3  min = 0
The max value is: 3
The minimum value is: 0
The sum is: 6
The average of the values is 2

C:\test>_
```

Here's the problem

C:\test\Extremes.java - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

Extremes.java

```
22
23          while (n != 0)
24          {
25              count++;
26              sum += n;
27              System.out.print("Enter number: ");  // Get additional inputs
28              n = scanner.nextInt();
29
30              if (n > max)
31              {
32                  max = n;
33              }
34              if (n < min)
35              {
36                  min = n;
37              }
38              System.out.println ("count = " + count + "  min = " + min);
39          }
40
```

Java source file | length : 1352 lines : 57 | Ln : 46 Col : 10 Sel : 0 | 0 | Dos\Windows | UTF-8 | INS

When the user enters 0 to terminate input, the program updates min with the value 0.

Solution

Don't update min and max if input is 0

*C:\test\Extremes.java - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

Extremes.java

```
23          while (n != 0)
24          {
25              count++;
26              sum += n;
27              System.out.print("Enter number: ");  // Get additional inputs
28              n = scanner.nextInt();
29              if (n != 0)
30              {
31                  if (n > max)
32                  {
33                      max = n;
34                  }
35                  if (n < min)
36                  {
37                      min = n;
38                  }
39              }
40              System.out.println ("count = " + count + "  min = " + min);
```

Java source file | length : 1383 lines : 59 | Ln : 42 Col : 9 Sel : 0 | 0 | Dos\Windows | UTF-8 | INS

Program Running

Command Prompt

```
C:\test>
C:\test>
C:\test>
C:\test>javac Extremes.java

C:\test>java Extremes
Enter number: 1
Enter number: 2
count = 1  min = 1
Enter number: 3
count = 2  min = 1
Enter number: 0
count = 3  min = 1
The max value is: 3
The minimum value is: 1
The sum is: 6
The average of the values is 2

C:\test>
```

Caution: We have corrected one error, but this program still has others. Don't view this as a correct solution for the Extremes project.

End of Section

Logical Operators

- Boolean expressions can use the following *logical operators*:

&&	Logical AND
||	Logical OR
!	Logical NOT

- All take boolean operands and produce boolean results.

Logical AND

- Logical AND and logical OR are *binary* operators.
  - Each operates on two operands.
- The *logical AND* expression

  `a && b`

  is true if *both* **`a`** and **`b`** are true.

  False otherwise.

- Two adjacent characters form a single operator.

Example

Boolean Expressions

```
if ( n1 < n2  &&  n2 < n3 )
{
    System.out.prinln("Numbers are in order\n");
}
```

Logical OR

- The *logical OR* expression

```
a || b
```

  is true if **a** is true
  or **b** is true
  or both are true.

- False only if *both* a and b are false.

- The "|" is the "vertical bar" character.
- Two adjacent characters form a single operator.

Example

Boolean Expressions

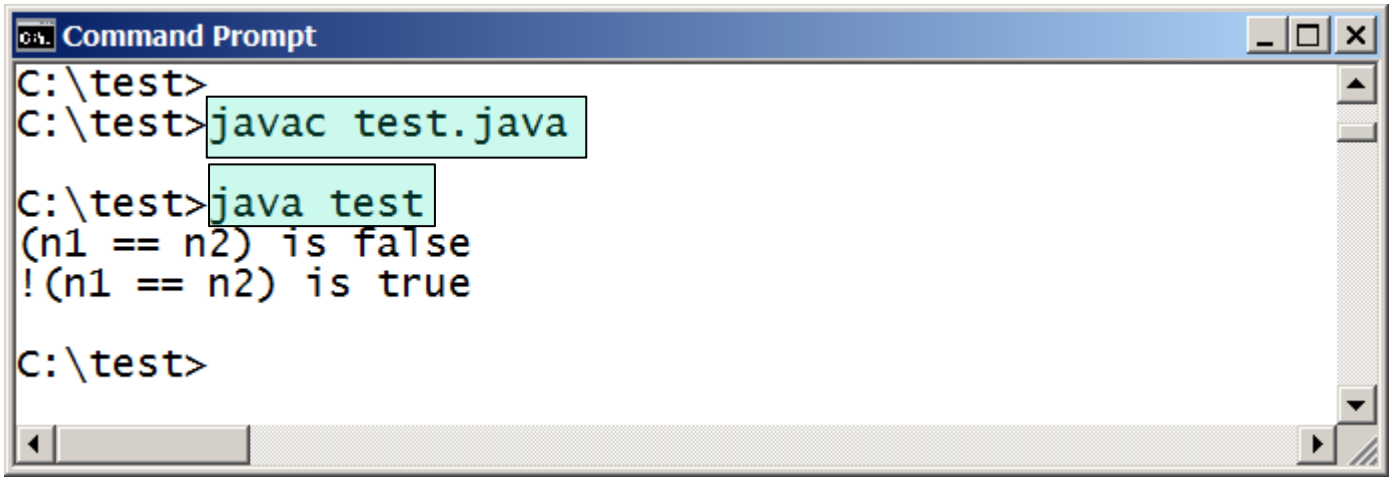
```
if ( n1 > 0   ||  n2 > 0 )
{
    System.out.println("Input is OK\n");
}
```

Logical NOT

- The *logical* NOT is a *unary* operator
  - It operates on one operand)
- If some boolean condition **`a`** is true, then **`!a`** is false;
- If **`a`** is false, then **`!a`** is true.
- **`a`** could be either a Boolean expression or a Boolean variable.

Example

```
class test
{
    public static void main(String[] args)
    {
        int n1 = 0;
        int n2 = 100;
        System.out.println("(n1 == n2) is " + (n1 == n2));
        System.out.println("!(n1 == n2) is " + !(n1 == n2));
    }
}
```



Logical Expressions

- Expressions that use logical operators can be combined to form complex conditions

```
if (total < MAX+5 && !found)
{
   System.out.println ("Processing…");
}
```

- All logical operators have lower precedence than the relational operators.
- Logical NOT has higher precedence than logical AND and logical OR.
- If you have trouble remembering these rules, use parentheses to force the order of evaluation.

Short-Circuited Operators

- The processing of logical AND and logical OR is "short-circuited"
- If the left operand is sufficient to determine the result, the right operand is not evaluated.

```
if (count != 0 && total/count > MAX)
{
    System.out.println ("Testing…");
    ...
}
```

If count is 0, you don't want to do the division!

Exercise

Rewrite the conditions below in valid Java syntax

- A. x and y are both less than 0

- B. x is equal to y but not equal to z

Answers

Rewrite the conditions below in valid Java syntax

- A. x and y are both less than 0

```
x < 0   &&   y < 0
```

- B. x is equal to y but not equal to z

```
x == y   &&   x != z
```

Comparing Data

- When comparing data using boolean expressions, it's important to understand the nuances of certain data types.
- Let's examine some key situations:
  - Comparing strings
  - Comparing characters
  - Comparing floating-point data

Comparing Strings

- Remember that in Java a character string is an *object.*
  - The == operator tests if the operands are the same string.
- The **`equals`** method can be called with strings to determine if two strings contain exactly the same characters in the same order.
  - What you normally want to know!

```
if (name1.equals(name2))
{
   System.out.println ("Same name");
}
```

Don't use the == operator for this!

Comparing Characters

- Unicode establishes a particular numeric value for each character, and therefore an ordering.
- We can use relational operators on character data based on this ordering.
- For example, the character `'+'` is less than the character `'J'` because it comes before it in the Unicode character set.

Comparing Characters

- In Unicode, the digit characters (0-9) are contiguous and in order.
- Likewise, the uppercase letters (A-Z) and lowercase letters (a-z) are contiguous and in order.

Characters	Unicode Values
0 – 9	48 through 57
A – Z	65 through 90
a – z	97 through 122

Comparing Floating Point Values

- Two floating point values are equal only if their underlying binary representations match exactly. Computations often result in slight differences that may be irrelevant
- In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal.
- You should rarely use the equality operator (**==**) when comparing two floating point values (**`float`** or **`double`**)

Comparing Floating Point Values

- To determine the equality of two floating point values, use the following technique:

```
if (Math.abs(f1 - f2) < TOLERANCE)
{
    System.out.println ("Essentially equal");
}
```

- If the difference between the two floating point values is less than the tolerance, they are considered to be equal

- The tolerance could be set to any appropriate level, such as 0.000001

The Conditional Operator

The Conditional Operator

- Syntax:

```
condition ? expression1 : expression2
```

- If **`condition`** is true, **`expression1`** is evaluated; if it is false, **`expression2`** is evaluated
- The value of the entire conditional operator is the value of the selected expression

The Conditional Operator

- The conditional operator is similar to an **`if-else`** statement, except that it is an expression that returns a value
- For example:

```
larger = ((num1 > num2) ? num1 : num2);
```

- If **`num1`** is greater than **`num2`**, then **`larger`** gets the value of **`num1`**
- Otherwise, **`larger`** gets the value of **`num2`**

The Conditional Operator

- It is never *necessary* to use the conditional operator. You can always do the same thing with **`if ... else`**, at the cost of somewhat more typing.

- For example, instead of

```
larger = ((num1 > num2) ? num1 : num2);
```

you could write

```
if (num1 > num2)
{
    larger = num1;
}
else
{
    larger = num2;
}
```

The switch Statement

The switch Statement

- The *switch statement* provides another way to decide which statement to execute next.

- It is a convenient shortcut when we need to compare a variable to several different values.

The switch Statement

- The *switch* statement evaluates an expression, then attempts to match the result to one of several possible *cases.*
- Each case contains a value and one or more statements
- The flow of control transfers to statement associated with the first case value that matches the expression.

The switch Statement

- The general syntax of a `switch` statement is:

switch and **case** are reserved words

```
switch ( expression )
{
    case value1 :
        statement-list1
    case value2 :
        statement-list2
    case value3 :
        statement-list3
    case  ...

}
```

If *expression* matches *value2*, control jumps to here

The switch Statement

- An example of a switch statement, assuming **`option`** is a character variable

```
switch (option)
{
    case 'A':
        aCount++;
        break;
    case 'B':
        bCount++;
        break;
    case 'C':
        cCount++;
        break;
}
```

The switch Statement

- Often a *break statement* is used as the last statement in each case's statement list.
- A *break* statement causes control to transfer to the end of the *switch* statement.
- If a *break* statement is not used, the flow of control will continue into the next case.
  - "fall through" feature of **`switch`**.
- *Normally* we only want to execute the statements associated with one case.

The switch Statement

- Another form of "fall through"

```
switch (option)
{
   case 'A':
   case 'B':
      abCount++;
      break;
   case 'C':
      cCount++;
      break;
}
```

The switch Statement

- A **`switch`** statement can have an optional *default case.*
- The default case has no associated value and simply uses the reserved word **`default`**
- If the default case is present, control will transfer to it if no other case value matches.
- If there is no default case, and no other value matches, control falls through to the statement after the switch.

The switch Statement

- The expression of a **`switch`** statement must result an **`int`** or a **`char`**
- It cannot be a **`boolean`** value, a floating point value (**`float`** or **`double`**), or another integer type (**`byte`**, **`short`**, or **`long`**).
- As of Java 7, a switch can also be used with strings.

```
//*******************************************************************
//  GradeReport.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a switch statement.
//*******************************************************************

import java.util.Scanner;

public class GradeReport
{
   //-----------------------------------------------------------------
   //  Reads a grade from the user and prints comments accordingly.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int grade, category;

      Scanner scanen = new Scanner (System.in);

      System.out.print ("Enter a numeric grade (0 to 100): ");
      grade = scaner.nextInt();

      category = grade / 10;

      System.out.print ("That grade is ");
```

continue

continue

```
        switch (category)
        {
            case 10:
                System.out.println ("a perfect score. Well done.");
                break;
            case 9:
                System.out.println ("well above average. Excellent.");
                break;
            case 8:
                System.out.println ("above average. Nice job.");
                break;
            case 7:
                System.out.println ("average.");
                break;
            case 6:
                System.out.println ("below average. You should see the");
                System.out.println ("instructor to clarify the material "
                                    + "presented in class.");
                break;
            default:
                System.out.println ("not passing.");
        }
    }
}
```

The switch Statement

- The implicit boolean condition in a **`switch`** statement is equality.
- You cannot perform relational checks with a **`switch`** statement.

Other Repetition Statements: The for Statement

The for Statement

- A *for statement* has the following syntax:

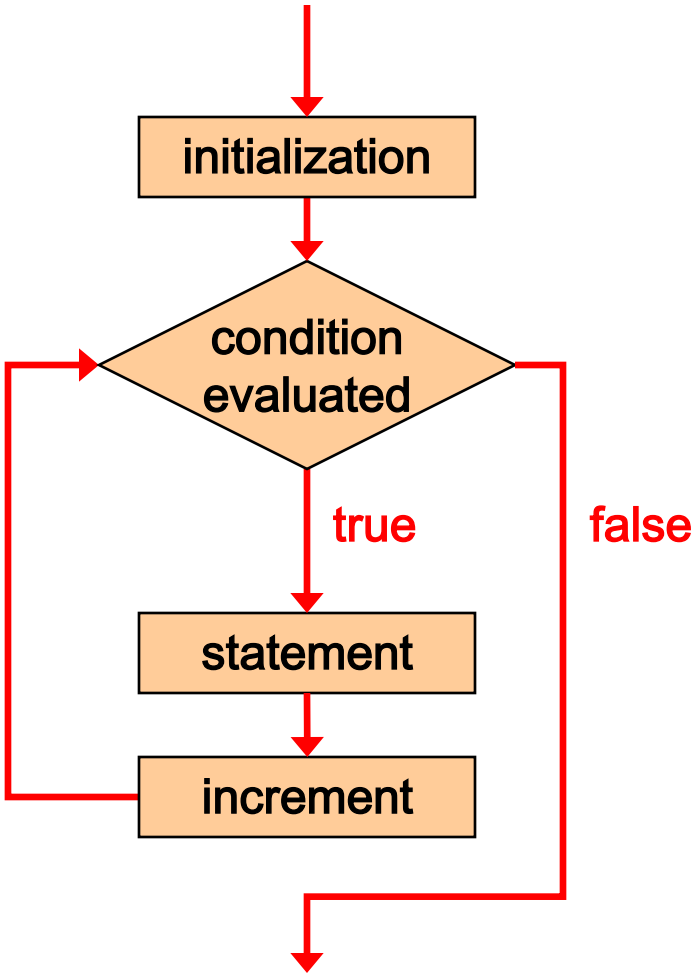
The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

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

The *increment* portion is executed at the end of each iteration

Logic of a `for` loop



The for Statement

- A **`for`** loop is functionally equivalent to the following **`while`** loop structure:

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

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

The `for` Statement

- An example of a **`for`** loop:

```
for (int count=1; count <= 5; count++)
{
   System.out.println (count);
}
```

- The initialization section can be used to declare a variable.
- Like a **`while`** loop, the condition of a **`for`** loop is tested prior to executing the loop body.
- Therefore, the body of a **`for`** loop will execute zero or more times.

  i.e. Might not be executed at all!

The for Statement

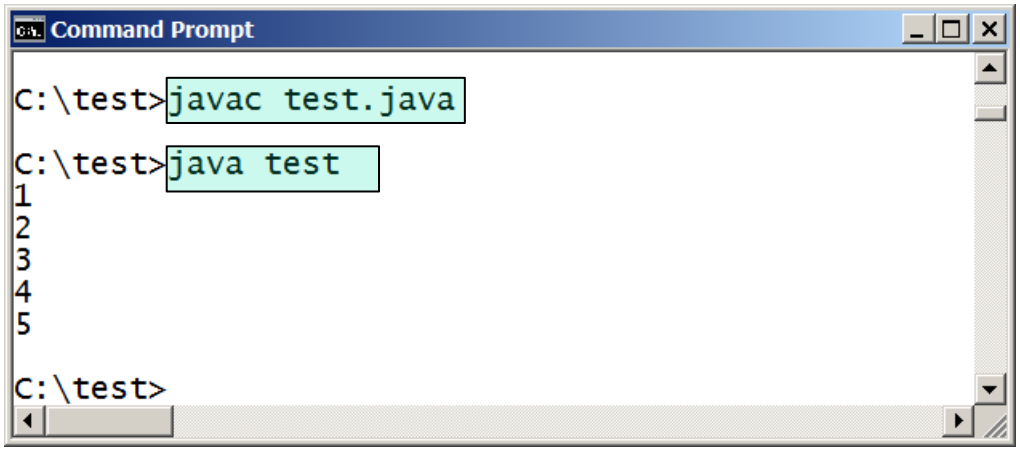
The "increment" section can perform *any* calculation.

```
for (int num=100; num > 0; num -= 5)
{
   System.out.println (num);
}
```

A **`for`** loop is well suited for executing statements a specific number of times that can be determined in advance.

Example

```
class test
{
    public static void main(String[] args)
    {
        for (int count=1; count <= 5; count++)
        {
            System.out.println (count);
        }
    }
}
```



The `for` Statement

- Each expression in the header of a **`for`** loop is optional.
- If the initialization is left out, no initialization is performed,

```
for(; val < 5; val ++)
```

- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop,

```
for(val =0; ; val ++)
```

- If the increment is left out, no increment operation is performed, **`for(;val<5;)`**

Recommendation: Don't do this!

```java
//*****************************************************************
//  Multiples.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a for loop.
//*****************************************************************

import java.util.Scanner;

public class Multiples
{
   //-----------------------------------------------------------------
   //  Prints multiples of a user-specified number up to a user-
   //  specified limit.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int PER_LINE = 5;
      int value, limit, mult, count = 0;

      Scanner scaner = new Scanner (System.in);

      System.out.print ("Enter a positive value: ");
      value = scaner.nextInt();
```

continue

continue

```
        System.out.print ("Enter an upper limit: ");
        limit = scan.nextInt();

        System.out.println ();
        System.out.println ("The multiples of " + value + " between " +
                        value + " and " + limit + " (inclusive) are:");

        for (mult = value; mult <= limit; mult += value)
        {
           System.out.print (mult + "\t");

           // Print a specific number of values per line of output
           count++;
           if (count % PER_LINE == 0)
           {
              System.out.println();
           }
        }
    }
}
```

Nested Loops

- Like nested `if` statements, loops can be nested.
  - The body of a loop can contain another loop.

- For each iteration of the outer loop, the inner loop iterates completely

Nested Loops

- How many times will the string **"Here"** be printed?

```
int count1 = 1;
while (count1 <= 10)
{
   int count2 = 1;
   while (count2 <= 5)
   {
      System.out.println ("Here " + count1 + " " + count2);
      count2++;
   }
   count1++;
}
```

10 * 5 = 50

Try it!

```
class test
{
    public static void main(String[] args)
    {
        int count1 = 1;
        while (count1 <= 10)
        {
           int count2 = 1;
           while (count2 <= 5)
           {
              System.out.println ("Here " + count1 + " " + count2);
              count2++;
           }
           count1++;
        }
    }
}
```

Nested Loop in Action

Command Prompt

```
Here 6 5
Here 7 1
Here 7 2
Here 7 3
Here 7 4
Here 7 5
Here 8 1
Here 8 2
Here 8 3
Here 8 4
Here 8 5
Here 9 1
Here 9 2
Here 9 3
Here 9 4
Here 9 5
Here 10 1
Here 10 2
Here 10 3
Here 10 4
Here 10 5

C:\test>_
```

```java
//*****************************************************************
//   Stars.java       Author: Lewis/Loftus
//
//   Demonstrates the use of nested for loops.
//*****************************************************************

public class Stars
{
   //-----------------------------------------------------------------
   //  Prints a triangle shape using asterisk (star) characters.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int MAX_ROWS = 10;

      for (int row = 1; row <= MAX_ROWS; row++)
      {
         for (int star = 1; star <= row; star++)
         {
            System.out.print ("*");
         }

         System.out.println();
      }
   }
}
```

Stars.java Running

Command Prompt

```
C:\test>
C:\test>javac Stars.java

C:\test>java Stars
*
**
***
****
*****
******
*******
********
*********
**********

C:\test>_
```

Exercise

Write a program that prints 10 stars on the diagonal.

```
*
 *
  *
   *
    *
     *
      *
       *
        *
         *
```

Loops with No Body

- The body associated with a conditional statement or loop can be empty.
  - This is because a *null* statement is syntactically valid.

- It means no statement is executed with the condition true.
  - Can be useful in some situations,
    but this is almost always a mistake.

- Examples:

```
if(value == 5);

while(value > 5);

for(int i =1; i<=5; i++);
```

Readings and Assignments

- Reading: Chapter 6.1-6.4, 5.1(b), 5.3
- Lab Assignment: Project 7

- Self-Assessment Exercises:
  - Self-Review Questions Section
    - SR5.4, 5.6, 5.7, 6.4, 6.7, 6.9, 6.13, 6.14, 6.15
  - After Chapter Exercises
    - EX 6.1, 6.2, 6.3, 6.6, 6.7, 6.11, 6.17