Some Practice Problems for the Java Exam

and Solutions for the Problems

The problems below are *not* intended to teach you how to program in Java. You should not attempt them until you believe you have mastered all the topics on the "Checklist" in the document entitled "C101 Java Study Guide".

There are 40 problems. The solutions for the problems are given at the end, after the statement of problem 40.

In all these problems, we'll assume that the following has been done in the appropriate places:

```
import java.util.Scanner;
Scanner scan = new Scanner(System.in);
```

1. What is the exact output of the program below. Indicate a blank line in the output by writing `blank line`.

```
public static void main(String[] args)
{
    int n = 4, k = 2;

    System.out.println(++n );
    System.out.println( n );

    System.out.println( n++ );
    System.out.println( n );

    System.out.println( -n );
    System.out.println( n );                    ...

    System.out.println( --n );
    System.out.println( n );

    System.out.println( n-- );             ...
    System.out.println( n );

    System.out.println( n + k );                ...
    System.out.println( n );
    System.out.println( k );                    ...
    System.out.println("" + n + " " + k );      ...

    System.out.println( n );                    ...
    System.out.println( "  " + n );             ...
    System.out.println( " n" );                 ...
    System.out.println( "\n" );                 ...

    System.out.println( " n * n = "); //CAREFUL!
    System.out.println( n * n );                ...
    System.out.println( 'n' );             ...
}
```

2. What is the output of the program below?

```
public static void main(String[] args)
{
    int n = 3;
    while (n >= 0)
    {
        System.out.println( n * n );
        --n;
    }

    System.out.println( n );

    while (n < 4)
        System.out.println( ++n );

    System.out.println( n );

    while (n >= 0)
        System.out.println( (n /= 2) );

}
```

3. What is the output of the program below?

```
public static void main(String[] args)
{
     int n;

    System.out.println( (n = 4) );
    System.out.println( (n == 4) );
    System.out.println( (n > 3) );
    System.out.println( (n < 4) );
    System.out.println( (n = 0) );
    System.out.println( (n == 0) );
    System.out.println( (n > 0) );
    System.out.println( (n == 0 && true) );
    System.out.println( (n == 0 || true) );
    System.out.println( (!(n == 2) ));

}
```

4. What is the output of the following program?

```
enum colorType {red, orange, yellow, green, blue, violet};

public static void main(String[] args)
{
    colorType shirt, pants;

    shirt = colorType.red;
    pants = colorType.blue;

    System.out.println("  " + shirt + "  " + pants );
}
```

5. What is the output when the following code fragment is executed?

```
int i = 5, j = 6, k = 7, n = 3;
System.out.println( i + j * k - k % n );
System.out.println( i / n );
```

6. What is the output when the following code fragment is executed?

```
int found = 0, count = 5;
if (!found || --count == 0)
    System.out.println( "danger" );
System.out.println( "count = " + count );
```

7. What is the output when the following code fragment is executed?

```
char [] title = {'T', 'i', 't', 'a', 'n', 'i', 'c'};

ch = title[1];
title[3] = ch;

System.out.println( title );
System.out.println( ch );
```

8. Suppose that the following code fragment is executed.

```
String message;

System.out.println( "Enter a sentence on the line below." );
message = scan.next();

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

a. Suppose that in response to the prompt, the interactive user types the following line and presses Enter:

```
Please go away.
```

What will the output of the code fragment look like?

b. Suppose that we modify the program and use `scan.nextLine()` instead of `scan.next()`, and that in response to the prompt, the interactive user types the following line and presses Enter:

```
Please stop bothering me.
```

What will the output of the code fragment look like?

9. Write a for loop that outputs all the *even* numbers between 0 and 20 in reverse order, from the largest to the smallest.

10. Write a for loop that outputs all the *odd* numbers between 0 and 20 in reverse order, from the largest to the smallest.

11. The nested conditional statement shown below has been written by an inexperienced Java programmer. The behavior of the statement is not correctly represented by the formatting.

```
if (n < 10)
    if (n > 0)
        System.out.println("The number is positive.");
else
    System.out.println("The number is ______________.");
```

a. What is the output of the statement if the variable n has the value 7 ? If n has the value 15 ? If n has the value -3 ?
b. Correct the syntax of the statement so that the logic of the corrected statement corresponds to the formatting of the original statement. Also, replace the blank with an appropriate word or phrase.
c. Correct the formatting of the (original) statement so that the new format reflects the logical behavior of the original statement. Also, replace the blank with an appropriate word or phrase.

12. The loop shown below has been written by an inexperienced Java programmer. The behavior of the loop is not correctly represented by the formatting.

```
int n = 10;

while (n > 0)
    n /= 2;
    System.out.println(n * n);
```

a. What is the output of the loop as it is written?
b. Correct the syntax of the loop so that the logic of the corrected loop corresponds to the formatting of the original loop. What is the output of the corrected loop?
c. Correct the formatting of the (original) loop so that the new format reflects the logical behavior of the original loop.

13. Remove all the unnecessary tests from the nested conditional statement below.

```
float income;

System.out.println("Enter your monthly income: ";
income = scan.nextFloat();

if (income < 0.0)
    System.out.println("You are going farther into debt every month.");
else if (income >= 0.0  &&  income < 1200.00)
    System.out.println("You are living below the poverty line.");
else if (income >= 1200.00  &&  income < 2500.00)
    System.out.println("You are living in moderate comfort.");
else if (income >= 2500.00)
    System.out.println("You are well off.");
```

14. Answer the questions below concerning the following fragment of code.

```
int n;
System.out.println("Enter an integer: ";
n = scan.nextInt();

if (n < 10)
    System.out.println("less than 10");

else if (n > 5)
    System.out.println("greater than 5");

else
```

```
        System.out.println("not interesting");
```

a. What will be the output of the fragment above if the interactive user enters the integer value 0 ?
b. What will be the output of the fragment above if the interactive user enters the integer value 15 ?
c. What will be the output of the fragment above if the interactive user enters the integer value 7 ?
d. What values for n will cause the output of the fragment above to be "not interesting"?

15. Rewrite the following code fragment so that it uses a "`do - while`" loop to accomplish the same task.

```
int n;

System.out.println("Enter a non-negative integer: ";
n = scan.nextInt();

while (n < 0)
{
    System.out.println("The integer you entered is negative.");
    System.out.println("Enter a non-negative integer: ";
    n = scan.nextInt();
}
```

16. In the code fragment below, the programmer has almost certainly made an error in the first line of the conditional statement.
a. What is the output of this code fragment as it is written?

```
int n = 5;
if (n == 0)
    System.out.println("n is zero.");
else
    System.out.println("n is not zero.");

    System.out.println("The square of n is " + n * n + ".");
```

17. What is the output when the following code fragment is executed?

```
int n, k = 5;
n = (100 % k == 0 ? k + 1 : k - 1);
System.out.println("n = " + n + "   k = " + k);
```

18. What is the output when the following code fragment is executed?

```
int   n;
float x = 3.8;
```

```
   n = (int) x;
   System.out.println("n = " + n);
```

19. What is the output when the following code fragment is executed? Rewrite the fragment to obtain an equivalent code fragment in which the body of the loop is a simple statement instead of a compound statement.

```
   int i = 5;
   while (i > 0)
   {
       --i;
       System.out.println(i);
   }
```

20. The following loop is an endless loop: when executed it will never terminate. What modification can be made in the code to produce the desired output?

```
   System.out.println("Here's a list of the ASCII values of all the upper"
                            + " case letters.");
   char letter = 'A';
   while (letter <= 'Z')
       System.out.println(letter + "  " + (int)letter);
```

21. Write a function named "`sumFromTo`" that takes two integer arguments, call them "`first`" and "`last`", and returns as its value the sum of all the integers between `first` and `last` inclusive. Thus, for example,

```
   System.out.println(sumFromTo(4,7));  // prints  22  because 4+5+6+7 = 22
   System.out.println(sumFromTo(-3,1)); // prints  -5  because
                                        // -3+(-2)+(-1)+0+1 = -5
   System.out.println(sumFromTo(7,4));  // prints  22  because 7+6+5+4 = 22
   System.out.println(sumFromTo(9,9));  // prints  9
```

22. Write a function named "`enough`" that takes one integer argument, call it "`goal`" and returns as its value the smallest positive integer n for which 1+2+3+. . . +n is at least equal to goal . Thus, for example,

```
   System.out.println(enough(9)); // will print  4  because  1+2+3+4 ≥ 9
                                  // but 1+2+3<9
   System.out.println(enough(21));// will print  6  because 1+2+ ...+6 ≥ 21
                                  // but 1+2+ . . . 5<21
   System.out.println(enough(-7));// will print  1  because  1 ≥ -7 and 1 is
                                  // the smallest positive integer
   System.out.println(enough(1)); // will print  1  because  1 ≥ 1 and 1 is
                                  // the smallest positive integer
```

DEFINITION: A positive integer d is called a divisor of an integer n if and only if the remainder after n is divided by d is zero. In this case we also say that " d divides n ", or that " n is divisible by d ". Here are some examples:

- 7 is a divisor of 35 ; that is, 35 is divisible by 7 .
- 7 is a not a divisor of 27 ; that is, 27 is not divisible by 7 .
- 1 is a divisor of 19 ; that is, 19 is divisible by 1 (in fact, 1 is a divisor of every integer n).
- 12 is a divisor of 0 ; that is, 0 is divisible by 12 .

In Java one can test the expression `n % d` to determine whether d is a divisor of n .
The greatest common divisor of a pair of integers m and n (not both zero) is the largest positive integer d that is a divisor of both m and n . We sometimes use the abbreviation "g.c.d." for "greatest common divisor. Here are some examples: 10 is the g.c.d. of 40 and 50 ; 12 is the g.c.d. of 84 and -132 ; 1 is the g.c.d. of 256 and 625 ; 6 is the g.c.d. of 6 and 42 ; 32 is the g.c.d. of 0 and 32 .

23. Write a function named "`gcd`" that takes two positive integer arguments and returns as its value the greatest common divisor of those two integers. If the function is passed an argument that is not positive (i.e., greater than zero), then the function should return the value 0 as a sentinel value to indicate that an error occurred. Thus, for example,

```
System.out.println(gcd(40,50));   // will print  10
System.out.println(gcd(256,625)); // will print  1
System.out.println(gcd(42,6));    // will print  6
System.out.println(gcd(0,32));    // will print  0  (even though  32  is
                                  // the g.c.d.)
System.out.println(gcd(10,-6));   // will print  0  (even though  2  is
                                  // the g.c.d.)
```

24. A positive integer n is said to be prime (or, "a prime") if and only if n is greater than 1 and is divisible only by 1 and n . For example, the integers 17 and 29 are prime, but 1 and 38 are not prime. Write a function named "isPrime" that takes a positive integer argument and returns as its value the integer 1 if the argument is prime and returns the integer 0 otherwise. Thus, for example,

```
System.out.println(isPrime(19));   // will print  1
System.out.println(isPrime(1));    // will print  0
System.out.println(isPrime(51));   // will print  0
System.out.println(isPrime(-13));  // will print  0
```

25. Write a function named "`digitName`" that takes an integer argument in the range from 1 to 9 , inclusive, and prints the English name for that integer on the computer screen. No newline character should be sent to the screen following the digit name. The function should not return a value. The cursor should remain on the same line as the name that has been printed. If the argument is not in the required range, then the function should print "digit error" without the quotation marks but followed by the newline character. Thus, for example,

the statement	`digitName(7);`	should print `seven` on the screen;
the statement	`digitName(0);`	should print `digit error` on the screen and place the cursor at the beginning of the next line.

26. Write a function named "`reduce`" that take one argument that is an array containing two positive integer values, and treats them as the numerator and denominator of a fraction, and reduces the fraction. That is to say, each of the two elements will be modified by dividing it by the greatest common divisor of the two integers. The function should return the value `false` (to indicate failure to reduce) if either of the two arguments is zero or negative, and should return the value `true` otherwise. Thus, for example, if m and n have been declared to be integer variables in a program, then

```
int[] fraction = {25, 15};
if (reduce(fraction))
    System.out.println("" + fraction[0] + '/' + fraction[1]);
else
    System.out.println("fraction error");
```

will produce the following output:

```
5/3
```

Note that the values of the fraction were modified by the function call. Similarly,

```
int[] fraction = {15, 50};
if (reduce(fraction))
    System.out.println("" + fraction[0] + '/' + fraction[1]);
       else
System.out.println("fraction error");
```

will produce the following output:

```
3/10
```

Here is another example.

```
int[] fraction = {25, 0};
if (reduce(fraction))
    System.out.println("" + fraction[0] + '/' + fraction[1]);
else
    System.out.println("fraction error");
```

will produce the following output:

```
fraction error
```

The function `reduce` is allowed to make calls to other functions that you have written.

27. Write a function named "`swapFloats`" that takes three arguments: a float array, and two indexes in this array. The function should interchange the values that are stored in the array at those indexes. The function should return no value. To take an example, if the following code fragment is executed

```
   float[] x = {5.8, 0.9);
   swapFloats (x, 0, 1);
   System.out.println("" + x[0] + "   " + x[1]);
```

then the output will be

```
   0.9   5.8
```

28. Write a function named "`sort3`" that takes an array containing three floating point arguments, call them "x" , "y" , and "z" , and modifies their values, if necessary, in such a way as to make true the following inequalities: $x \leq y \leq z$. The function should return no value. To take an example, if the following code fragment is executed

```
   float[] a = {3.2, 5.8, 0.9};
   sort3 (a);
   System.out.println(""a[0] + "   " + a[1] + "   " + a[2]);
```

then the output will be

```
   0.9   3.2   5.8
```

The function `sort3` is allowed to make calls to other functions that you have written.

29. Write a function named "`reverse`" that takes as argument an array of floating point values. The function must reverse the order of the values in the array. Thus, for example, if the array that's passed to the function looks like this:

0	1	2	3	4
5.8	2.6	9.0	3.4	7.1

then when the function returns, the array will have been modified so that it looks like this:

0	1	2	3	4
7.1	3.4	9.0	2.6	5.8

The function should not return any value.

30. Write a function named "`sum`" that takes as argument an array of floating point values. The function should return as its value the sum of the floating point values in the array. Thus, for example, if the array that's passed to the function looks like this:

```
  0     1     2     3     4
5.8 | 2.6 | 9.0 | 3.4 | 7.1
```

then the function should return the value 27.9 as its value.

31. Write a function named "`locationOfLargest`" that takes as argument an array of integer values. The function should return as its value the subscript of the cell containing the largest of the values in the array. Thus, for example, if the array that's passed to the function looks like this:

```
0    1    2    3    4
58 | 26 | 90 | 34 | 71
```

then the function should return the integer 2 as its value. If there is more than one cell containing the largest of the values in the array, then the function should return the smallest of the subscripts of the cells containing the largest values. For example, if the array that's passed to the function is

```
0    1    2    3    4    5    6
58 | 26 | 91 | 34 | 70 | 91 | 88
```

then the largest value occurs in cells 2 and 5 , so the function should return the integer value 2 .

32. Write a function named "`locationOfTarget`" that takes as its arguments the following:
(1) an array of integer values;
(2) an integer "target value".
The function should determine whether the given target value occurs in any of the cells of the array, and if it does, the function should return the subscript of the cell containing the target value. If more than one of the cells contains the target value, then the function should return the largest subscript of the cells that contain the target value. If the target value does not occur in any of the cells, then the function should return the sentinel value -1 . Thus, for example, if the target value that's passed to the function is 34 and the array that's passed to the function looks like this:

```
0    1    2    3    4    5    6
58 | 26 | 91 | 34 | 70 | 34 | 88
```

then the target value occurs in cells 3 and 5 , so the function should return the integer value 5 .

33. Write a function named "`rotateRight`" that takes as argument an array of floating point values. The function should shift the contents of each cell one place to the right, except for the contents of the last cell, which should be moved into the cell with subscript 0 . Thus, for example, if the array passed to the function looks like this:

```
  0     1     2     3     4
5.8 | 2.6 | 9.1 | 3.4 | 7.0
```

then when the function returns, the array will have been changed so that it looks like this:

```
  0     1     2     3     4
7.0 | 5.8 | 2.6 | 9.1 | 3.4
```

The function should not return a value.

34. Write a function named "`shiftRight`" that takes as its arguments the following:
(1) an array of floating point values;
(2) an integer, call it "left", that tells the leftmost cell of the part of the array to be shifted;
(3) an integer, call it "right", that tells the rightmost cell of the part of the array to be shifted;
(4) a positive integer, call it "distance" that tells how many cells to shift by.
The function should make sure that left is less than or equal to right, and that distance is greater than zero. If either of these conditions fails, the function should return the value false to indicate an error. Otherwise it should shift by distance cells the contents of the array cells with subscripts running from left to right . Thus, for example, if the array passed to the function looks like this:

```
  0     1     2     3     4     5     6     7      8     9    10    ....
5.8 | 2.6 | 9.1 | 3.4 | 7.0 | 5.1 | 8.8 | 0.3 | -4.1 | 8.0 | 2.7 | etc.
```

and if left has the value 3 , right has the value 7 , and distance has the value 2 , then the function should shift the contents of cells 3 , 4 , 5 , 6 , and 7 to the right by 2 cells, so that when the function returns, the array will have been changed so that it looks like this:

```
  0     1     2     3     4     5     6     7     8     9    10   ....
5.8 | 2.6 | 9.1 | ??? | ??? | 3.4 | 7.0 | 5.1 | 8.8 | 0.3 | 2.7 | etc.
```

The question marks in cells 3 and 4 indicate that we don't care what numbers are in those cells when the function returns. Note that the contents of cells 8 and 9 have changed, but the contents of cell 10 is unchanged. The function need not take any precautions against the possibility that the cells will be shifted beyond the end of the array (the calling function should be careful not to let that happen).

35. Write a function named "`subtotal`" takes as argument an array of floating point values. The function should replace the contents of each cell with the sum of the contents of all the cells in the original array from the left end to the cell in question. Thus, for example, if the array passed to the function looks like this:

```
  0     1     2     3     4
```

```
|5.8 | 2.6 | 9.1 | 3.4 | 7.0|
```

then when the function returns, the array will have been changed so that it looks like this:

```
  0     1     2      3      4
|5.8 | 8.4 | 17.5 | 20.9 | 27.9|
```

because 5.8 + 2.6 = 8.4 and 5.8 + 2.6 + 9.1 = 17.5 and so on. Note that the contents of cell 0 are not changed. The function should not return a value.

36. Write a function named "`concatenate`" that copies the cells of one array into a larger array, and then copies the cells of another array into the larger array just beyond the contents of the first array. The contents of the cells will be integers. The arguments will be as follows:
(1) the first array that will be copied;
(2) the second array that will be copied;
(3) the large array into which all copying will be performed;
If the function discovers that the number of cells in the large array is not large enough to hold all the numbers to be copied into it, then the function should return false to indicate failure. Otherwise it should return true . The function should not alter the contents of the first two arrays. To take an example, if the first two arrays passed to the function look like this:

```
 0    1    2    3    4    5    6                 0    1    2    3
|58 | 26 | 91 | 34 | 70 | 34 | 88|     and     |29 | 41 | 10 | 66|
```

then, provided the size of the large array is at least 11, the large array should look like this when the function returns:

```
 0    1    2    3    4    5    6    7    8    9   10
|58 | 26 | 91 | 34 | 70 | 34 | 88 | 29 | 41 | 10 | 66|
```

37. Write a function named "`numberOfMatches`" that compares the initial parts of two character arrays to see how many pairs of cells match before a difference occurs. For example, the call

```
System.out.println(numberOfMatches("Bookclub", "Bookcase"));
```

will print the value 5.

38. Write a function named "`eliminateDuplicates`" that takes an array of integers in random order and eliminates all the duplicate integers in the array. The function should take as argument an array of integers. The function should return a value indicating how many unique values were found in the array. Here is an example. Suppose the array passed to the function is as shown below, of size 11.

```
   0    1    2    3    4    5    6    7    8    9    10
 58 | 26 | 91 | 26 | 70 | 70 | 91 | 58 | 58 | 58 | 66
```

Then the function should alter the array so that it looks like this:

```
   0    1    2    3    4    5    6    7    8    9    10
 58 | 26 | 91 | 70 | 66 | ?? | ?? | ?? | ?? | ?? | ??
```

and it should return the value 5. The question marks in the cells after the 5th cell indicate that it does not matter what numbers are in those cells when the function returns.

39. Write a function called "`isOrdered`" that takes one parameter representing an array of float values and returns true if the values of the array are in ascending order, and false otherwise. For example, if the array contains the values

```
3  4  12  18  19  20  20  24  27
```

then the function should return true. Note that duplicate values are allowed.
However, if the array contains the values

```
42  13  27  84  2  12  42  95
```

the function should return false.

40. Write an entire Java program that reads a positive integer entered by an interactive user and then prints out all the positive divisors of that integer in a column and in decreasing order. The program should allow the user to repeat this process as many times as the user likes. Initially, the program should inform the user about how the program will behave. Then the program should prompt the user for each integer that the user wishes to enter.

The program may be terminated in any of two ways. One way is to have the program halt if the user enters an integer that is negative or zero. In this case the user should be reminded with each prompt that the program can be terminated in that way. Alternatively, after an integer has been entered and the divisors have been printed, the program can ask the user whether he/she wishes to enter another integer. In this case, when the user accidentally enters a zero or negative integer to have its divisors calculated, the program should inform the user that the input is unacceptable and should allow the user to try again (and again!).

Here is an illustration of how the program and the interactive user might interact. The user's responses to the program are shown in bold italics.

```
This program is designed to exhibit the positive divisors of
positive integers supplied by you.  The program will repeatedly
prompt you to enter a positive integer.  Each time you enter a
positive integer, the program will print all the divisors of your
integer in a column and in decreasing order.

Please enter a positive integer: 36
36
18
12
9
6
4
3
2
1

Would you like to see the divisors of another integer (Y/N)? y
Please enter a positive integer: -44

-44 is not a positive integer.
Please enter a positive integer: 0

0 is not a positive integer.
Please enter a positive integer: 109
109
1

Would you like to see the divisors of another integer (Y/N)? m

Please respond with Y (or y) for yes and N (or n) for no.
Would you like to see the divisors of another integer (Y/N)? n
```

Answers

1. The output would be as shown below.

```
5
5
5
6
-6
6
5
5
5
4
6
4
2
4 2
4
  4
 n
blank line
blank line
 n * n = 16
n
```

2. The output would be as shown below. The program contains an endless loop.

```
9
4
1
0
-1
0
1
2
3
4
4
2
1
0
0 [endlessly]
```

3. The output would be as shown below.

```
4
true
true
false
0
true
false
true
true
true
```

4. The output would be as shown below.

```
  red  blue
```

5. The output would be as shown below.

```
46
1
```

6. The output would be as shown below, because when it is discovered that "!found" is true, the second half of the OR expression is not evaluated, and thus count is not decremented.

```
danger
count = 5
```

7. The output would be as shown below. The 'a' in "Titanic" has been changed to an 'i'.

```
Titinic
i
```

8. a. The output would be as shown below. The line in italics would be typed by the interactive user.

```
Enter a sentence on the line below.
Please go away.
Please
```

b. The output would be as shown below. The line in italics would be typed by the interactive user.

```
Enter a sentence on the line below.
Please stop bothering me.
Please stop bothering me.
```

9. Here is the fragment of code:

```
    for (int i = 20; i >= 0, i -= 2)
        System.out.println(i);
```

10 Here is the fragment of code:

```
    for (int i = 19; i >= 0, i -= 2)
        System.out.println(i);
```

11 a. The original statement is formatted in such a way that it appears that the "`else`" clause of the statement is the alternative to the "`if (n < 10)`" case, but in fact, the C or Java compiler will treat the "`else`" clause as the alternative to the "`if (n > 0)`" clause. If n has the value 7 , the output will be "`The number is positive`". If n has the value 15 , then there will be no output. If n has the value -3 then the output will be "`The number is` ____________".

b. If we want the statement to behave according the logic that's suggested by the formatting of the original statement, we can write

```
if (n < 10)
{
    if (n > 0)
        System.out.println("The number is positive.");
}
else
    System.out.println("The number is greater than 10.");
```

c. If we want the statement to be formatted so as to reflect the actual logic of the original statement, we can write

```
if (n < 10)
    if (n > 0)
        System.out.println("The number is positive.");
    else
        System.out.println("The number is negative.");
```

12 a. Since there are no braces surrounding the last two lines of the code, the compiler treats only the statement "`n /= 2;`" as the body of the loop. Thus n will successively assume the values 5, 2, 1, and 0, at which point the loop will exit and the output statement will print 0 . The values 5, 2, and 1 will not be printed.

b. If we want the statement to behave according the logic that's suggested by the formatting of the original statement, we can put braces around the last two lines of code to make a compound statement as the body of the loop:

```
int n = 10;

while (n > 0)
{
    n /= 2;
    System.out.println(n * n);
}
```

In this case the output of the loop will be

```
25
4
1
0
```

c. If we want the statement to be formatted so as to reflect the actual logic of the original statement, we can write

```
int n = 10;

while (n > 0)
   n /= 2;

System.out.println(n * n);
```

13. The conditional statement should be modified as follows:

```
if (income < 0.0)
    System.out.println("You are going farther into debt every month.");
else if (income < 1200.00)
    System.out.println("You are living below the poverty line.");
else if (income < 2500.00)
    System.out.println("You are living in moderate comfort.");
else
    System.out.println("You are well off.");
```

14 a. The output will be `"less than 10"`.
b. The output will be `"greater than 15"`.
c. The output will be `"less than 10"`.
d. There is no value for n that will cause the output to be `"not interesting"`. That part of the code can never be executed!

15. Using do - while... in this situation makes the code a little simpler.

```
int n;

do
{
    System.out.println("Enter a non-negative integer: ";
    n= scan.nextInt();

    if (n < 0)
        System.out.println("The integer you entered is negative.");
}
while (n < 0);
```

16 a. The output of the code fragment as it is written will be

```
n is not zero.
The square of n is 25.
```

The reason for this is that the "if" part assigns the value 0 to n , and the value returned by that assignment statement is 0 , so this is treated as "false", which causes the alternative statement to be executed. But even though the program prints "`n is not zero.`", in fact, n does have the value zero after the conditional statement is finished.

17. The output of the code fragment is as follows:

```
n = 4   k = 5
```

18. The output of the code fragment is as follows:

```
n = 3
```

The fractional part of the floating point value is discarded when the value is cast to integer type.

19. The output of the code fragment is as follows:

```
4
3
2
1
0
```

The loop can be rewritten as

```
int i = 5;
while (i > 0)
    System.out.println(--i);
```

20. The loop can be modified as follows:

```
while (letter <= 'Z')
{
    System.out.println(letter + "  " + (int)letter);
   ++letter;
}
```

21.

```
/****************** S U M   F R O M   T O **********************

DESCRIPTION:  Computes and returns the sum of all the integers
              between "first" and "last" inclusive.

PARAMETERS:

  first, last The two "endpoints" of the sequence of integers to be
              summed.

RETURNS:      Returns the sum of all the integers from "first" to
              "last".  For example, if first is 9 and last is 12
              then the function will return 42 ( = 9+10+11+12).
              If first is 11 and last is 8, then the function
              will return 38 ( = 11+10+9+8).  If first is 5 and
              last is 5, the function will return 5.

ALGORITHM:    If first <= last, the addition begins at first and
              goes up to last.  If, instead, first > last, then the
              addition begins at first and goes down to last.

AUTHORs:      D. Vrajitoru & W. Knight

*******************************************************************/

static int sumFromTo (int first, int last)
{
    int i, partialSum = 0;

    if (first <= last)
        for (i = first; i <= last; ++i)
            partialSum += i;

    else
        for (i = first; i >= last; --i)
            partialSum += i;

    return partialSum;
}
```

22.

```
/*************************  E N O U G H  *************************

DESCRIPTION:  Computes and returns the smallest positive integer n
              for which 1+2+3+...+n equals or exceeds the value of
              "goal".

PARAMETER:

  goal        The integer which 1+2+3+...+n is required to meet or
              exceed.

RETURNS:      Returns the smallest positive integer n  for which
              1+2+3+...+n equals or exceeds "goal".  For example, if
              goal has the value 9, then the function will return 4
              because 1+2+3+4 >= 9 but 1+2+3 < 9.  If goal has a value
              less than or equal to 1, then the function will return
              the value 1.

ALGORITHM:    First the value n = 1 is tried to see if 1 >= goal.
              If that is not true, then successively larger values
              of n are added to a summing variable until that sum
              reaches or exceeds goal.

AUTHORs:      D. Vrajitoru & W. Knight

******************************************************************/

static int enough (int goal)
{
   int n = 1, sum = 1;

   while (sum < goal)
      sum += ++n;

   return n;
}
```

23. There is a well-known algorithm called "Euclid's Algorithm" for computing the g.c.d. of two positive integers. The second of the two solutions below employs that algorithm. The first solution employs a somewhat more straightforward search process, in which we simply try one integer after another, starting with the smaller of the two arguments, until we find an integer that divides both. For the purposes of the Java exam, the first solution is acceptable (even though it is far less efficient than the solution that uses Euclid's Algorithm).

```
/**********************  G C D (VERSION 1)  ************************

DESCRIPTION:  Computes and returns the greatest common divisor
              (g.c.d.) of the arguments passed to it.

PARAMETERS:

  a , b       The integers whose g.c.d. will be computed.

RETURNS:      If either argument is less than or equal to zero, the
              value zero will be returned as a sentinel to indicate
              an error.  If both arguments are strictly positive,
              then their g.c.d. will be returned.  Thus, for
              example, if parameter a has the value 28 and b has the
              value 70, then the function will return the value 14.

ALGORITHM:    If both arguments are positive, then the smaller of the
              two arguments is tested to see whether it is a divisor
              of both arguments.  If it is not, then successively
              smaller integers are tried.  If no common divisor larger
              than 1 is found, then the loop will automatically stop
              with trialDivisor having the value 1 because 1 is a
              divisor of every positive integer.

AUTHORs:      D. Vrajitoru & W. Knight

*******************************************************************/

int gcd (int a, int b)
{
    if (a <= 0  ||  b <= 0)  // a parameter is not positive
        return 0;            // exit and return the error sentinel

    int trialDivisor;
    trialDivisor = ( a <= b  ?  a  :  b ); // set it to the smaller

    while (a % trialDivisor != 0  ||  b % trialDivisor != 0)
        --trialDivisor;

    return trialDivisor;
}
```

A version of the previous algorithm that uses the Euclidean algorithm is shown below.

```
/*********************  G_C_D (VERSION 2)  ***********************

DESCRIPTION:  Computes and returns the greatest common divisor
              (g.c.d.) of the arguments passed to it.

PARAMETERS:

  a , b       The integers whose g.c.d. will be computed.

RETURNS:      If either argument is less than or equal to zero, the
              value zero will be returned as a sentinel to indicate
              an error.  If both arguments are strictly positive,
              then their g.c.d. will be returned.  Thus, for
              example, if parameter a has the value 28 and b has the
              value 70, then the function will return the value 14.

ALGORITHM:    If both arguments are positive, then Euclid's algorithm
              for the g.c.d. is employed.  The first integer is
              divided by the second, and then the second is divided
              by the remainder, and then the first remainder is
              divided by the second, and so on until a remainder of 0
              is obtained.
              The g.c.d. will be the divisor that produced 0
              remainder.

AUTHOR:       W. Knight

*******************************************************************/

static int gcd (int a, int b)
{
    if (a <= 0  ||  b <= 0)  // a parameter is not positive
        return 0;             // exit and return the error sentinel

    int remainder = a % b;   // Get remainder when a is divided by b.

    while (remainder != 0)
    {
        a = b;
        b = remainder;
        remainder = a % b;
    }

    return b;  // Return the divisor that produced a remainder of 0.
}
```

24.

```
/************************  I S   P R I M E  **********************

DESCRIPTION:  Determines whether an integer is prime.

PARAMETER:

  n           The integer to be examined to see whether it is prime.

RETURNS:      1 if the integer n is prime;  otherwise it returns 0.

ALGORITHM:    If n is less than or equal to 1, then it is not prime.
              If n is greater than 1, then trial divisors, starting
              with 2 and going no farther than n-1 are examined to
              determine whether they divide n.  If no divisor less
              than n is found, then n is prime.

AUTHORS:      D. Vrajitoru & W. Knight

*******************************************************************/


static int isPrime (int n)
{
    if (n <= 1)
        return 0; // n cannot be prime if n <= 1.

    int trialDivisor = 2;

    while (trialDivisor < n  &&  n % trialDivisor != 0)
        ++trialDivisor;

    // When the loop exits, one of two conditions must be satisfied:
    // either trialDivisor will have reached the value  n -- which
    // means that  n  is prime or else  n  will be divisible by
    // trialDivisor, in which case  n  will not be prime.  The only
    // exception to this is when n is 2, in which case n is prime.

    if (trialDivisor == n) // n must be prime
        return 1;
    else
        return 0; // n is not prime.
}
```

25.

```
/*********************  D I G I T   N A M E  **********************

DESCRIPTION:  _Prints the English name of an integer from 1 to 9.

PARAMETER:

  n           The integer whose English name will be printed.

RETURNS:      Void (no value).

ALGORITHM:    A switch statement selects the appropriate word.
              If "n" is not in the range 1,2,3,...,9, then an
              error phrase is printed.

AUTHORS:      D. Vrajitoru & W. Knight

*******************************************************************/


static void digitName (int digitValue)
{
    switch (digitValue)
    {
        case 1  : System.out.println("one");    break;
        case 2  : System.out.println("two");    break;
        case 3  : System.out.println("three");  break;
        case 4  : System.out.println("four");   break;
        case 5  : System.out.println("five");   break;
        case 6  : System.out.println("six");    break;
        case 7  : System.out.println("seven");  break;
        case 8  : System.out.println("eight");  break;
        case 9  : System.out.println("nine");   break;
        default : System.out.println("digit error");
    }
}
```

26.

```
/*************************  R E D U C E  ***********************

DESCRIPTION:  Reduces a positive fraction to lowest terms.

PARAMETERS:

  fraction    Array containing the numerator and denominator of the
              fraction to be reduced.

RETURNS:      1 if the arguments are both positive, 0 otherwise.

ALGORITHM:    If both values are positive, then each of them is
              divided by their greatest common divisor.

AUTHORS:      D. Vrajitoru & W. Knight

*******************************************************************/

static boolean reduce (int[] fraction)
{
   if (fraction[0] <= 0  ||  fraction[1] <= 0)
      return false;

   else
   {
      int common = gcd (fraction[0], fraction[1]);
      fraction[0] /= common;
      fraction[1] /= common;
      return true;
   }
}
```

27.

```
/*********************  S W A P   F L O A T S  *********************

DESCRIPTION:  Interchanges the values of the two floating point
              elements of an array passed to it.

PARAMETERS:

  x, a, b     The first is a float array, and the other two the indexes
              of the elements to be interchanged.

RETURNS:      void (no value).

ALGORITHM:    Stores the value of parameter a in a temporary location,
              then copies the value in x[b] into x[a], and finally copies
              the stored original value of x[a] into x[b].

AUTHORS:      D. Vrajitoru & W. Knight

*******************************************************************/

static void swapFloats (float[] x, int a, int b)
{
   float temp = x[a];
   x[a] = x[b];
   x[b] = temp;
}
```

28. Here is one solution among many possible. It uses the `swapFloats` function of problem 27.

```
/*************************  S O R T 3  ***************************
DESCRIPTION:  Sorts an array containing three values passed to it into
              increasing order.

PARAMETERS:
  x           The array containing the three floating point numbers to be
              sorted.

RETURNS:      void (no value).

ALGORITHM:    First x[0], x[1], and x[2] are compared to see if they are
              already in increasing order. If they are, no changes are
              made.
              If x[0] <= x[1] but x[1] > x[2], then x[1] and x[2] are
              swapped, and then x[0] and the new value of x[1] are
              compared and put in correct order.
              If x[0] > x[1], then x[0] and x[1] are swapped.
              Next it is necessary to discover where x[2] belongs in
              relation to x[0] and x[1].  If x[1] <= x[2], nothing more
              needs doing.
              If, instead, x[2] < x[1], then x[1] and x[2] are swapped
              and then x[0] and the new x[1] are compared and put in
              order.

AUTHORS:      D. Vrajitoru & W. Knight
*******************************************************************/

static void sort3 (float[] x)
{
    float temp;

    if (x[0] <= x[1]  &&  x[1] <= x[2]) // the values are already in
        return;                         // order; do nothing

    if (x[0] <= x[1]) // then x[1] > x[2] (or we'd have stopped above)
    {
        swapFloats (x, 1, 2); // After this call, x[1] < x[2] and
                              // x[0] <= x[2] are true but we don't know
        if (x[0] > x[1])      // how x[0] and x[1] compare.
            swapFloats (x, 0, 1);  // Now x[0] < x[1] <= x[2] must be
    }                              // true.

    else // If neither of the above is true, then x[1] < x[0] is true
    {
        swapFloats (x, 0, 1);  // After this call, x[0] < x[1] is true.

        if (x[1] <= x[2])  // the values are now in correct order;
            return;        // do nothing
            else // it must be the case that x[1] > x[2]
            {
                swapFloats (x[1], x[2]);  // After this call, x[1] <= x[2] is
                if (x[0] > x[1])          // true.
                    swapFloats (x, 0, 1);
            }
        }
    }
```

29. The following function calls the swapFloats function given in problem 27.

```
/************************  R E V E R S E  *************************

DESCRIPTION:  Reverses the order of the objects in an array.

PARAMETERS:

  a           The array of floating point numbers whose objects will
              be reversed.

RETURNS:      Void (no value).

ALGORITHM:    One array index starts at the left end of the array and
              moves to the right, while another starts at the right
              end and moves to the left.  Objects in the cells
              indicated by these two indexes are swapped.  The
              process ends when the two indexes meet or cross each
              other.

AUTHORS:      D. Vrajitoru & W. Knight

*******************************************************************/


static void reverse (float[] a)
{
    int n = a.length;
    int i = 0, j = n - 1;

    while (i < j)
        swapFloats (a, i++, j--);
}
```

30.

```
/*****************************  S U M  ****************************

DESCRIPTION:  Calculates and returns the sum of the numbers in an array.

PARAMETERS:

  a           The array of floating point numbers to be summed

RETURNS:      The sum  a[0] + a[1] + . . . + a[length-1].

ALGORITHM:    A summing variable is initialized to zero and then each
              floating point value in the array is added in turn.

AUTHORS:      D. Vrajitoru & W. Knight

*******************************************************************/

static float sum (float[] a,)
{
    int   n = a.length;
    float sum = 0.0;
    int   i;

    for (i = 0; i < n; ++i)
        sum += a[i];

    return sum;
}
```

31. Here is the simplest solution.

```
/************* L O C A T I O N   O F   L A R G E S T *************

DESCRIPTION:  Finds the index of the largest number in an array.

PARAMETERS:

  a           An array of integers to be scanned.

RETURNS:      The index of the cell containing the largest integer.
              If the largest integer appears in more than one cell,
              then the index of the leftmost cell where it appears
              will be returned.

ALGORITHM:    A variable that will hold the index of the largest
              integer is initialized to zero, and we pretend to have
              scanned that cell and found that it contains the
              largest integer seen "so far".  Then successive cells
              are examined and compared with the cell containing the
              largest integer so far.  When a cell containing a new
              largest integer is encountered, the variable that holds
              the index of the largest integer seen so far is
              modified to the new index.

AUTHORS:      D. Vrajitoru & W. Knight

******************************************************************/

static int locationOfLargest (int[] a)
{
    int n = a.length;
    int best = 0; // Location of the largest so far.
    int i;

    for (i = 1; i < n; ++i) // Start comparing at the second cell.
        if (a[i] > a[best])
            best = i;

    return best;
}
```

32. Here is the simplest solution. It searches from right to left and quits when (if) if finds the target.

```
/**************  L O C A T I O N   O F   T A R G E T  **************

DESCRIPTION:  Finds the index of the cell (if any) where a "target"
              integer is stored.

PARAMETERS:

  a           An array of integers to be scanned.

  target      The integer that we hope to find in some cell of
              array a.

RETURNS:      The index of the cell containing the integer "target"
              provided there is such a cell.  If there is not, then
              the function returns -1 as a sentinel value.
              If the target integer appears in more than one cell,
              then the index of the rightmost cell where it appears
              will be returned.

ALGORITHM:    The length of the array "n" is used to scan backward across
              the array, looking for a cell containing a copy of
              "target".  If a copy is found, the search is halted and
              the index of that cell is returned.  If no such cell is
              found, "n" will run off the left end of the array and
              wind up with value -1.

AUTHORS:      D. Vrajitoru & W. Knight

*******************************************************************/


static int locationOfTarget (int[] a, int target)
{
    int n = a.length;

    --n;  // Make n "point" to the last cell of the array.

    while (n >= 0  &&  a[n] != target) // Search right to left
        --n;

    return n;  // Returns -1 if target is not in array a[].
}
```

33.

```
/******************  R O T A T E   R I G H T  ********************

DESCRIPTION:  Shifts the contents of array cells one cell to the
              right, with the last cell's contents moved to the left
              end.

PARAMETERS:

  a           The floating point array to be modified.

RETURNS:      void (no value).

ALGORITHM:    The object in the right-most cell is copied to a
              temporary location, and then the object each cell to
              the left of the last cell is copied to its immediate
              right neighbor;  the process moves from right to left.
              Finally, the object in the temporary location is copied
              to the leftmost cell.

AUTHORS:      D. Vrajitoru & W. Knight

******************************************************************/

static void rotateRight (float[] a)
{
    int   n = a.length;
    float temp = a[n-1];  // Hold the contents of the last cell.
    int i;

    for (i = n - 1;  i >= 1;  --i)
        a[i] = a[i-1];

    a[0] = temp;
}
```

34.

```
/*********************  S H I F T   R I G H T  ********************

DESCRIPTION:  Shifts the contents of some subarray in an array to the
              right by a specified number of cells.

PARAMETERS:

  a           The floating point array to be modified.

  left        The index of the leftmost cell of the subarray to be
              shifted.

  right       The index of the rightmost cell of the subarray.

  distance    The number of cells by which the subarray will be
              shifted.

RETURNS:      true provided left <= right and distance > 0;  returns
              false if either of these conditions is violated.

ALGORITHM:    An index variable is initialized to "point" to the
              rightmost cell of the subarray to be shifted;  another
              index variable is initialized to point to the cell to
              which the rightmost object will be copied.  Then the
              copying takes place and the indexes are moved to the
              left (decremented by 1).  The occurs repeatedly until
              the object in the leftmost cell of the subarray has
              been copied.

AUTHORS:      D. Vrajitoru & W. Knight

*******************************************************************/

static boolean shiftRight (float a[], int left, int right, int distance)
{
   if (left > right  ||  distance <= 0)
      return false;

   int i = right,        // points to the cell to be shifted
       j = i + distance; // points to the receiving cell
   while (i >= left)
   {
      a[j] = a[i];
      --i;
      --j;
   }
   return true;
}
```

35.

```
/************************  S U B T O T A L  **********************

DESCRIPTION:  Replaces each number in an array with the sum of all
              the numbers up to that location in the original array.

PARAMETERS:

  a           The floating point array to be modified.

RETURNS:      void (no value).

ALGORITHM:    Starting with cell 1 and moving right, the number in
              each cell is replaced by the sum of that number and the
              sum that's now in the cell just to the left.

AUTHORS:      D. Vrajitoru & W. Knight

*******************************************************************/

static void subtotal (float[] a)
{
    int n = a.length;
    int i;

    for (i = 1; i < n; ++i)
        a[i] += a[i-1];
}
```

36.

```
/**********************  C O N C A T E N A T E  *******************

DESCRIPTION:  Copies numbers from two arrays into a third array.  The
              numbers from the second array are placed to the right
              of the numbers copied from the first array.

PARAMETERS:

  a, b        The arrays from which numbers will be copied into "c".

  c           The array that will receive the copied numbers.

RETURNS:      true, provided the capacity of c is at least equal to the
              number of numbers that are to be copied from arrays a
              and b;  false is returned if this condition fails.

ALGORITHM:    If c has adequate capacity, then the leftmost m cells
              of array a are copied to the leftmost m cells of c, and
              then the leftmost n cells of b are copied into the
              n cells of c lying just to the right of the first m
              cells.

AUTHORS:      D. Vrajitoru & W. Knight

******************************************************************/

static boolean concatenate (int[] a[], int[] b, int[] c)
{
    int m = a.length, n = b.length, p = c.length;

    if (m + n > p)
        return false;

    int i, j;

    for (i = 0; i < m; ++i)
        c[i] = a[i];

    for (j = 0; j < n; ++j)
        c[i++] = b[j];       // Increment i and j after each assignment

    return true;
}
```

37.

```
/****************  N U M B E R   O F   M A T C H E S  ***************

DESCRIPTION:  Compares two strings to determine how many of the initial
characters match.

PARAMETERS:

  a, b       The strings to be compared.


RETURNS:     The number of non-NUL initial matches in the two
             arrays. For example, if the arrays contain "boasted" and
             "boats" (with NUL following the final 'd' and 's'
             respectively), then 3 will be returned because "boa"
             matches "boa".

ALGORITHM:   The characters are compared pairwise, starting at cell
             0, until a NUL is reached or a pair of characters does
             not match.

AUTHORS:     D. Vrajitoru & W. Knight

*******************************************************************/

static int numberOfMatches (String a, String b)
{
    int n = a.length(), m = b.length();

    int i = 0; // i keeps track of how many pairs of cells match

    while (i < n && i < m  &&  a.charAt(i) == b.charAt(i))
        ++i;

    return i;
}
```

38.

```
/***********  E L I M I N A T E   D U P L I C A T E S  **************

DESCRIPTION:  Examines an array of integers and eliminates all
              duplication of values.  The distinct integers are all
              moved to the left part of the array.

PARAMETERS:

  a           The integer array to be modified.

RETURNS:      void (no value).

ALGORITHM:    At all times during the algorithm, an integer keeps
              track of the last cell of the subarray containing all
              the distinct integers that have been found and moved to
              that subarray.  Another integer moves along the
              unexamined portion of the array.  Each time a new
              integer is encountered, it is checked against all the
              integers in the subarray of distinct integers to see
              whether it should be added to that subarray.

AUTHORS:      D. Vrajitoru & W. Knight

*******************************************************************/


static int eliminateDuplicates (int[] a[])
{
    int n = a.length;
    int lastUnique = 0; // Keeps track of the end of the subarray of
                        // integers known to be all different.
    int i;
    for (i = 1; i < n; ++i) // Start i at the second cell (number 1).
    {
        // Determine whether a[i] is already present among the integers
        // in cells a[0],...,a[lastUnique].
        int j = 0;
        while (j <= lastUnique  &&  a[i] != a[j])
            ++j;
        if (j > lastUnique)           // then a[i] is not present
            a[++lastUnique] = a[i];  // earlier, so put a[i] into the
                                     // list of all different integers.
    }
    return lastUnique + 1; // Return the number of distinct integers in
                           // the processed array.
}
```

39.

```
/*********************  I S   O R D E R E D  *********************

DESCRIPTION:  Examines an array of floats and returns true if the values
are ordered in ascending order, allowing for duplicate values.  Otherwise
it returns false.

PARAMETERS:

  a           The integer array to be verified.

RETURNS:      true if the array is sorted in ascending order,
              false otherwise.

ALGORITHM:    The algorithm scans the array from left to right and
              compares each element with the one to the right of it. If
              the element is larger than the next one, then it returns
              false. If it reaches the end of the array without having to
              return false, then it returns true.

AUTHORS:      D. Vrajitoru & W. Knight

*******************************************************************/


static boolean isOrdered (int[] a)
{
    int n = a.length;
    int i;

    for (i = 0; i < n-1; ++i) // Stop before the last element.
        // Determine whether a[i] and a[i+1] are in the right order.
        if (a[i] > a[i+1])  // not in order
            return false;

    return true; // If we made it here, the array is in order.

}
```

40.

```
/******************************************************************

PROGRAMMERS:    D. Vrajitoru & W. Knight

DESCRIPTION:    This program prompts an interactive user to enter
                positive integers, and it prints all the positive
                divisors of each positive integer entered.

NOTE:           If the user enters one or more non-numeric characters
                when prompted for an integer, the program will go
                into an endless loop.

******************************************************************/

import java.util.Scanner;

class Main()
{

    static Scanner scan;

    /************************  M A I N  ***************************/

    public static void main(String[] args)
    {
        int usersInteger;

        scan = new Scanner(System.in);

        printInitialExplanation();

        do
        {
            usersInteger = getNumberFromUser ();

            printDivisors (usersInteger);
        }
        while (userWishesToRepeat());
    }
```

```
/******* P R I N T   I N I T I A L   E X P L A N A T I O N ********

DESCRIPTION:   This function prints some text on the screen.

PARAMETERS:    None.

RETURNS:       Void (no value).

WRITTEN BY:    D. Vrajitoru & W. Knight

*******************************************************************/

static void printInitialExplanation (void)
{
    System.out.println("\n\nThis program is designed to exhibit the"
                       + "positive ");
    System.out.println("divisors of positive integers supplied by "
                       + "you.  The");
    System.out.println("program will repeatedly prompt you to enter a");
    System.out.println("positive integer.  Each time you enter a "
                       + "positive");
    System.out.println("integer, the program will print all the divisors"
                       + " of");
    System.out.println("your integer in a column and in decreasing "
                       + "order.\n\n";
}
```

```
/************  G E T   N U M B E R   F R O M   U S E R  *************

DESCRIPTION:   This function prompts an interactive user for a
               positive integer, reads the integer, and returns it in
               the variable passed to this function.

PARAMETER:
   n           The positive integer supplied by the user.  This is a
               reference parameter.  It is intended that the argument
               passed through this parameter will receive a new
               value.

RETURNS:       Void (no value).

ALGORITHM:     The user is prompted for the positive integer.  If the
               integer entered by the user is zero or negative, the
               user will be informed that the input is not positive
               and will be given another chance to enter valid data.
               The function will not exit until the user has entered
               valid data.

WRITTEN BY:    D. Vrajitoru & W. Knight
*******************************************************************/
static int getNumberFromUser ()
{
    int n;
    do
    {
        System.out.print("Please enter a positive integer: ");
        n = scan.nextInt();
        if (n <= 0)
            System.out.println("\n" + n + " is not a positive "
                                   + "integer.");
    }
    while (n <= 0);
    return n;
}
// --------------------------------------------------------------------
// COMMENT:  In the function  "getNumberFromUser", if the
// interactive user enters a non-numeric character in response to the
// prompt for a positive integer, then the program will go into
// an endless loop.
```

```
/******************  P R I N T   D I V I S O R S  *****************

DESCRIPTION:   This function prints all the divisors of a positive
               integer in a column and in decreasing order.

PARAMETER:

   n           The positive integer whose divisors will be printed.

RETURNS:       void (no value).

ALGORITHM:     First the value of n is printed.  Then, starting with
               the next possible smaller divisor (n/2), each integer
               down to 1 is tested to see if it is a divisor of n,
               and every divisor is printed.

WRITTEN BY:    D. Vrajitoru & W. Knight

*******************************************************************/

static void printDivisors (int n)
{
    System.out.println(n);

    int trialDivisor;

    for (trialDivisor = n / 2;  trialDivisor >= 1;  --trialDivisor)
        if (n % trialDivisor == 0)
            System.out.println(trialDivisor);

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

```
   /**********  U S E R   W I S H E S   T O   R E P E A T  *************

   DESCRIPTION:   This function determines whether an interactive user
                  wishes to enter another positive integer.

   PARAMETERS:    None.

   RETURNS:       true if the user wishes to repeat, false if not.

   ALGORITHM:     The user is asked to type Y for "yes, I wish to repeat"
                  or N for "no, I do not".  The user's response is then
                  read from the keyboard, and if it is neither Y nor N
                  (the lower case versions of these characters are also
                  accepted), then the user is asked to re-enter a response.

   WRITTEN BY:    D. Vrajitoru & W. Knight

   ********************************************************************/

   static boolean userWishesToRepeat ()
   {
       String response;

       System.out.println("Would you like to see the divisors of another";
       System.out.println(" integer (Y/N)? ";
       response = scan.next();

       while (response != "Y"  &&  response != "y"  &&  response != "N"
                                                 &&  response != "n")
       {
           System.out.println("\nPlease respond with Y (or y) for yes and N"
                              + " (or n)";
           System.out.println(" for no.");

           System.out.println("Would you like to see the divisors of "
                              + "another integer (Y/N)? ");
           response = scan.next();

       }

       if (response == "Y"  ||  response == "y")
           return true;
       else
           return false;
   }
} // end of the class
```