NESTED FOR LOOPS

Control structures in Java are statements that contain statements. In particular, control structures can contain control structures. You've already seen several examples of `if` statements inside loops, and one example of a `while` loop inside another `while`, but any combination of one control structure inside another is possible. We say that one structure is nested inside another. You can even have multiple levels of nesting, such as a `while` loop inside an `if` statement inside another `while` loop. The syntax of Java does not set a limit on the number of levels of nesting. As a practical matter, though, it's difficult to understand a program that has more than a few levels of nesting.

Nested `for` loops arise naturally in many algorithms, and it is important to understand how they work. Let's look at a couple of examples. First, consider the problem of printing out a multiplication table like this one:

```
 1   2   3   4   5   6   7   8   9  10  11  12
 2   4   6   8  10  12  14  16  18  20  22  24
 3   6   9  12  15  18  21  24  27  30  33  36
 4   8  12  16  20  24  28  32  36  40  44  48
 5  10  15  20  25  30  35  40  45  50  55  60
 6  12  18  24  30  36  42  48  54  60  66  72
 7  14  21  28  35  42  49  56  63  70  77  84
 8  16  24  32  40  48  56  64  72  80  88  96
 9  18  27  36  45  54  63  72  81  90  99 108
10  20  30  40  50  60  70  80  90 100 110 120
11  22  33  44  55  66  77  88  99 110 121 132
12  24  36  48  60  72  84  96 108 120 132 144
```

The data in the table are arranged into 12 rows and 12 columns. The process of printing them out can be expressed in a pseudocode algorithm as

```
for each rowNumber = 1, 2, 3, ..., 12:
   Print the first twelve multiples of rowNumber on one line
   Output a carriage return
```

The first step in the `for` loop can itself be expressed as a `for` loop. We can expand "Print the first twelve multiples of `rowNumber` on one line" as:

```
for N = 1, 2, 3, ..., 12:
   Print N * rowNumber
```

so a refined algorithm for printing the table has one `for` loop nested inside another:

```
for each rowNumber = 1, 2, 3, ..., 12:
   for N = 1, 2, 3, ..., 12:
      Print N * rowNumber
   Output a carriage return
```

We want to print the output in neat columns, with each output number taking up four spaces. This can be done using formatted output with format specifier `%4d`. Assuming that `rowNumber` and `N` have been declared to be variables of type int, the algorithm can be expressed in Java as

```
for ( rowNumber = 1;  rowNumber <= 12;  rowNumber++ ) {
   for ( N = 1;  N <= 12;  N++ ) {
               // print in 4-character columns
      System.out.printf(  "%4d",  N  *  rowNumber  );    //  No
carriage return !
   }
   System.out.println();   // Add a carriage return at end of
the line.
}
```

This section has been weighed down with lots of examples of numerical processing. For our next example, let's do some text processing. Consider the problem of finding which of the 26 letters of the alphabet occur in a given string. For example, the letters

that occur in "Hello World" are D, E, H, L, O, R, and W. More specifically, we will write a program that will list all the letters contained in a string and will also count the number of different letters. The string will be input by the user. Let's start with a pseudocode algorithm for the program.

```
Ask the user to input a string
Read the response into a variable, str
Let count = 0  (for counting the number of different letters)
for each letter of the alphabet:
   if the letter occurs in str:
      Print the letter
      Add 1 to count
Output the count
```

Since we want to process the entire line of text that is entered by the user, we'll use `TextIO.getln()` to read it. The line of the algorithm that reads "for each letter of the alphabet" can be expressed as "`for (letter='A'; letter<='Z'; letter++)`". But the body of this `for` loop needs more thought. How do we check whether the given letter, `letter`, occurs in `str`? One idea is to look at each character in the string in turn, and check whether that character is equal to `letter`. We can get the `i`-th character of `str` with the function call `str.charAt(i)`, where `i` ranges from 0 to `str.length() - 1`.

One more difficulty: A letter such as 'A' can occur in `str` in either upper or lower case, 'A' or 'a'. We have to check for both of these. But we can avoid this difficulty by converting `str` to upper case before processing it. Then, we only have to check for the upper case letter. We can now flesh out the algorithm fully:

```
Ask the user to input a string
Read the response into a variable, str
Convert str to upper case
Let count = 0
```

```
for letter = 'A', 'B', ..., 'Z':
    for i = 0, 1, ..., str.length()-1:
        if letter == str.charAt(i):
            Print letter
            Add 1 to count
            break  // jump out of the loop, to avoid counting
letter twice
Output the count
```

Note the use of `break` in the nested `for` loop. It is required to avoid printing or counting a given letter more than once (in the case where it occurs more than once in the string). The `break` statement breaks out of the inner `for` loop, but not the outer `for` loop. Upon executing the `break`, the computer continues the outer loop with the next value of `letter`. You should try to figure out exactly what `count` would be at the end of this program, if the `break` statement were omitted.

Here is the complete program and an applet to simulate it:

```
/**
 * This program reads a line of text entered by the user.
 * It prints a list of the letters that occur in the text,
 * and it reports how many different letters were found.
 */

public class ListLetters {

   public static void main(String[] args) {

      String str;  // Line of text entered by the user.
      int count;   // Number of different letters found in str.
      char letter; // A letter of the alphabet.

      TextIO.putln("Please type in a line of text.");
      str = TextIO.getln();
```

```
            str = str.toUpperCase();

            count = 0;
            TextIO.putln("Your    input    contains    the    following
        letters:");
            TextIO.putln();
            TextIO.put("   ");
            for ( letter = 'A'; letter <= 'Z'; letter++ ) {
                int i;  // Position of a character in str.
                for ( i = 0; i < str.length(); i++ ) {
                    if ( letter == str.charAt(i) ) {
                        TextIO.put(letter);
                        TextIO.put(' ');
                        count++;
                        break;
                    }
                }
            }

            TextIO.putln();
            TextIO.putln();
            TextIO.putln("There   were   "   +   count   +   "   different
        letters.");

           } // end main()

        } // end class ListLetters
```

In fact, there is actually an easier way to determine whether a given letter occurs in a string, `str`. The built-in function `str.indexOf(letter)` will return `-1` if `letter` does **not** occur in the string. It returns a number greater than or equal to zero if it does occur. So, we could check whether `letter` occurs in `str` simply by checking "`if (str.indexOf(letter) >= 0)`". If we used this technique in

the above program, we wouldn't need a nested `for` loop. This gives you a preview of how subroutines can be used to deal with complexity.

Source : http://math.hws.edu/javanotes/c3/s4.html