2IP67.5 Instruction 5

5.1 Fixed point

Puzzle: fill in digits on the dots, such that the following sentence is true: *This sentence contains exactly ... times the digit 0, ... times the digit 1, ... times the digit 2, ... times the digit 3, ... times the digit 4, ... times the digit 5, ... times the digit 6, ... times the digit 7, ... times the digit 8, ... times the digit 9.*

1. First try to find a solution by yourself. Of course all digits in the sentence count with the totals, also the digits in the numbers you fill in.

   Now we are going to solve this by computer using a so-called *fixed point* technique.

2. Let us represent a "filling-in" of the sentence by an array of 10 integers, each element of the array representing the number filled in in the corresponding position of the sentence. Write a method that takes as a parameter such an array of 10 ints and prints the sentence above, with the numbers in the array filled in in the order of the array. So when the parameter is the array containing `{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}`, the method should print

   ```
   This sentence contains exactly 0 times the digit 0, 1 times the digit 1, 2 times the digit 2,
   3 times the digit 3, 4 times the digit 4, 5 times the digit 5, 6 times the digit 6, 7 times
   the digit 7, 8 times the digit 8, 9 times the digit 9.
   ```

   which is not true, by the way.

3. * Perfectionists might modify the method that it prints "1 time" instead of "1 times". Consult the Sun tutorial on "Conditional operator" to achieve this in a compact way.

4. To see whether a filling-in is correct, write a method *count* that counts how many times each digit occurs in the sentence when you would fill it in and returns an array with the resulting counts. So the example filling-in of part 2 would result in the array `{2, 2, 2, 2, 2, 2, 2, 2, 2, 2}`. The header of this method should be

   ```
   int[] count(int[] a)
   ```

   Make sure that this method *does not change the array* `a`. Create a new array in the body and return this array.

   You may assume that the numbers are always less than 10. Print a warning when a number exceeds 9 and continue as if it was 9. See part 8 for an improvement.

   Test it by printing both the sentence and the result of *count*. You can print the contents of a (short) array a by first applying `Arrays.toString(a)`, or write a method that gives a nice output.

5. Write a method that checks whether a filling-in is correct, i.e., the result of count of this filling-in is exactly equal to the filling-in. You can use the method `Arrays.equals(a, b)` to check whether two arrays are equal (element-wise). Make sure `import java.util.*;` is present in your program.

   Note that the assignment `a=b`, where a and b are both arrays, doesn't copy the contents of b into a, it just makes a refer to the same array as b. If you need to copy the contents of an array, use a loop, or use `a = Arrays.copyOf(b, b.length);` (the second parameter gives the number of elements to copy).

6. Now the idea is to use the result of count as a new attempt to the solution and continue until a *fixed point* has been reached, i.e., a filling in a for which $count(a) = a$. So write a loop that starts with an initial filling-in, applies count, and uses the result as the parameter to count again, etc. Stop when the fixed point has been reached. Display intermediate results.

   Try different initial fillings-in. How many solutions do you find?

7. This fixed point technique doesn't always find a solution. There are two sources of trouble: (1) infinitely growing numbers and (2) oscillation, i.e., the same sequence of fillings-in is repeatedly visited, but none of them is a solution.

   (a) Can you see why trouble source 1 is not possible?

   (b) * Write a refinement that terminates the program when an oscillation of length 2 occurs.

8. When a digit occurs more than 9 times, the method count has to be refined. Use the method `int[] digitFreq(int[] n)` presented in the lecture that counts the digits in the decimal representation of n, returning an array of length 10 with the frequencies of the digits of n, where the i'th element contains the frequency in n of digit i.