**[Java](http://www.dcon.com.br/jd.comment/articles/notes-java/index.html): Example - Factorial**

Factorial n (usually written *n!*) is the product of all integers up to and including n (1 \* 2 \* 3 \* ... \* n). This problem is often (inappropriately) used as a programming example, especially to show how to write recursive functions.

**Recursive solution is a bad example**

Writing a recursive factorial function is a mistake because

* The recursive solution's memory usage is O(N) instead of O(1).
* It's more complicated for students.
* It's inappropriate when the iterative solution is better.

Recursion is a great tool, but using it where it's not appropriate doesn't set a great example.

**Range and precision problems**

Even the iterative solution has problems because the large numbers that factorial generates cannot be represented in the limited range of integers, or limited precision of floating-point numbers. Programs using these primitive types are very dangerous because they will produce garbage results without comment.

**Solution with java.math.BigInteger**

The proper solution is to use [java.Math.BigInteger](http://www.dcon.com.br/jd.comment/articles/notes-java/data/numbers/60factorial.html). Here is a program with int and BigInteger solutions.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | // numbers/Factorial.java - Computes factorial  // Fred Swartz - 2003-11-02  import java.math.BigInteger;  public class Factorial {  public static void main(String[] args) {    //-- BigInteger solution.  BigInteger n = BigInteger.ONE;  for (int i=1; i<=20; i++) {  n = n.multiply(BigInteger.valueOf(i));  System.out.println(i + "! = " + n);  }    //-- int solution (BAD IDEA BECAUSE ONLY WORKS TO 12).  int fact = 1;  for (int i=1; i<=20; i++) {  fact = fact \* i;  System.out.println(i + "! = " + fact);  }  }  } |

**Sample output**

|  |  |
| --- | --- |
| **BigInteger output** | **int output** |
| 1! = 1  2! = 2  3! = 6  4! = 24  5! = 120  6! = 720  7! = 5040  8! = 40320  9! = 362880  10! = 3628800  11! = 39916800  12! = 479001600  13! = 6227020800  14! = 87178291200  15! = 1307674368000  16! = 20922789888000  17! = 355687428096000  18! = 6402373705728000  19! = 121645100408832000  20! = 2432902008176640000 | 1! = 1  2! = 2  3! = 6  4! = 24  5! = 120  6! = 720  7! = 5040  8! = 40320  9! = 362880  10! = 3628800  11! = 39916800  12! = 479001600  13! = 1932053504 BAD  14! = 1278945280 BAD  15! = 2004310016 BAD  16! = 2004189184 BAD  17! = -288522240 BAD  18! = -898433024 BAD  19! = 109641728 BAD  20! = -2102132736 BAD |

|  |  |  |
| --- | --- | --- |
| An int produces pure garbage after 12, and long only makes it to 20. The shame of integer arithmetic is that it doesn't stop when the values are bad -- it just produces garbage by throwing away bits in the result that don't fit in an int (32 bits) or long (64-bits). BigInteger values are limited only by the amount of memory. |  |  |