```java
import components.naturalnumber.NaturalNumber;

/**
 * Program to test arrays, references, and arrays of references.
 *
 * @author Gage Farmer
 *
 */
public final class ArraysAndReferences {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private ArraysAndReferences() {
    }

    /**
     * Computes the product of the {@code NaturalNumber}s in the given array.
     *
     * @param nnArray
     *            the array
     * @return the product of the numbers in the given array
     * @requires nnArray.length > 0
     * @ensures <pre>
     * productOfArrayElements =
     *    [nnArray[0] * nnArray[1] * ... * nnArray[nnArray.length-1]]
     * </pre>
     */
    private static NaturalNumber productOfArrayElements(
            NaturalNumber[] nnArray) {
        assert nnArray != null : "Violation of: nnArray is not null";
        assert nnArray.length > 0 : "Violation of: nnArray.length > 0";

        NaturalNumber total = new NaturalNumber2(1);

        for (int i = 0; i < nnArray.length; i++) {
            total.multiply(nnArray[i]);
        }

        return total;
    }

    /**
     * Replaces each element of {@code nnArray} with the partial product of all
     * the elements in the incoming array, up to and including the current
     * element.
     *
     * @param nnArray
     *            the array
     * @updates nnArray
     * @requires nnArray.length > 0
     * @ensures <pre>
     * for all i: integer where (0 <= i < nnArray.length)
     *   (nnArray[i] = [#nnArray[0] * #nnArray[1] * ... * #nnArray[i]])
     * </pre>
     */
    private static void computePartialProducts(NaturalNumber[] nnArray) {
        assert nnArray != null : "Violation of: nnArray is not null";
        assert nnArray.length > 0 : "Violation of: nnArray.length > 0";

        NaturalNumber[] copy = nnArray;

        for (int i = 1; i < nnArray.length; i++) {
            nnArray[i].multiply(copy[i - 1]);
        }

    }

    /**
     * Creates and returns a new array of {@code NaturalNumber}s, of the same
     * size of the given array, containing the partial products of the elements
     * of the given array.
     *
     * @param nnArray
     *            the array
     * @return the array of partial products of the elements of the given array
     * @requires nnArray.length > 0
     * @ensures <pre>
     * partialProducts.length = nnArray.length  and
     *  for all i: integer where (0 <= i < partialProducts.length)
     *    (partialProducts[i] = [nnArray[0] * nnArray[1] * ... * nnArray[i]])
     * </pre>
     */
    private static NaturalNumber[] partialProducts(NaturalNumber[] nnArray) {
        assert nnArray != null : "Violation of: nnArray is not null";
        assert nnArray.length > 0 : "Violation of: nnArray.length > 0";

        // TODO - fill in body

        /*
         * This line added just to make the program compilable. Should be
         * replaced with appropriate return statement.
         */
        return null;
    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments
     */
    public static void main(String[] args) {
        SimpleWriter out = new SimpleWriter1L();

        /*
         * Initialize an array of NaturalNumbers with values 1 through 5.
         */
        NaturalNumber[] array = new NaturalNumber[5];
        NaturalNumber count = new NaturalNumber2(1);
        for (int i = 0; i < array.length; i++) {
            array[i] = new NaturalNumber2(count);
            count.increment();
        }
        /*
         * Compute and output the product of the numbers in the array (should be
         * 42!, i.e., the factorial of 42).
         */
        NaturalNumber product = productOfArrayElements(array);
        out.println(product);

        computePartialProducts(array);
        for (int i = 0; i < array.length; i++) {
            out.print(array[i].toString() + " ");
        }

        out.close();
    }

}
```