```java
import components.naturalnumber.NaturalNumber;
import components.naturalnumber.NaturalNumber2;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 * Program with implementation of {@code NaturalNumber} secondary operation
 * {@code root} implemented as static method.
 *
 * @author Put your name here
 *
 */
public final class NaturalNumberRoot {

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

    /**
     * Updates {@code n} to the {@code r}-th root of its incoming value.
     *
     * @param n
     *            the number whose root to compute
     * @param r
     *            root
     * @updates n
     * @requires r >= 2
     * @ensures n ^ (r) <= #n < (n + 1) ^ (r)
     */
    public static void root(NaturalNumber n, int r) {
        /**
         * NaturalNumber instance to represent the lower bound of the guess, the
         * upper bound of the guess, the initial guess for the
```

```java
root, integer
         * value 1, integer value 2, and difference between upper and lower
         * bounds.
         **/
        NaturalNumber lowEnough = new NaturalNumber2(0);
        NaturalNumber tooHigh = new NaturalNumber2(n);
        NaturalNumber guess = new NaturalNumber2(n);
        NaturalNumber one = new NaturalNumber2(1);
        NaturalNumber two = new NaturalNumber2(2);
        NaturalNumber dif = new NaturalNumber2(n);
        boolean loop = true;
        /**
         * While the difference between the upper and lower bounds of the guess
         * range is greater than or equal to 1 and loop is true
         **/
        while (dif.compareTo(one) >= 0 && loop) {
            NaturalNumber guessPow = new NaturalNumber2(guess);
            guessPow.power(r);

            if (guessPow.compareTo(n) == 0) {
                n.copyFrom(guess);
                loop = false;
            } else if (guessPow.compareTo(n) < 0) {
                lowEnough.copyFrom(guess);
                guess.add(tooHigh);
                guess.divide(two);
            } else {
                tooHigh.copyFrom(guess);
                guess.add(lowEnough);
                guess.divide(two);
            }

            if (guess.compareTo(lowEnough) == 0
                    || guess.compareTo(tooHigh) == 0) {
                n.copyFrom(guess);
                loop = false;
            }
            /**
             * Calculate the new value of the difference between the upper and
             * lower bounds of the guess range
             **/
            NaturalNumber lowEnough2 = new NaturalNumber2(lowEnough);
            NaturalNumber tooHigh2 = new NaturalNumber2(tooHigh);
            lowEnough2.add(tooHigh2);
            lowEnough2.divide(two);
            guess.copyFrom(lowEnough2);

            dif.copyFrom(tooHigh);
            dif.subtract(lowEnough);

        }
    }

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

        final String[] numbers = { "0", "1", "13", "1024", "189943527", "0",
                "1", "13", "4096", "189943527", "0", "1", "13", "1024",
                "189943527", "82", "82", "82", "82", "82", "9", "27", "81",
                "243", "143489073", "2147483647", "2147483648",
                "9223372036854775807", "9223372036854775808",
                "618970019642690137449562111",
                "162259276829213363391578010288127",
                "170141183460469231731687303715884105727" };
        final int[] roots = { 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 15, 15, 15, 15, 15,
                2, 3, 4, 5, 15, 2, 3, 4, 5, 15, 2, 2, 3, 3, 4, 5, 6 };
        final String[] results = { "0", "1", "3", "32", "13782", "0", "1", "2",
                "16", "574", "0", "1", "1", "1", "3", "9", "4", "3", "2", "1",
                "3", "3", "3", "3", "3", "46340", "46340", "2097151", "2097152",
                "4987896", "2767208", "2353973" };

        for (int i = 0; i < numbers.length; i++) {
            NaturalNumber n = new NaturalNumber2(numbers[i]);
            NaturalNumber r = new NaturalNumber2(results[i]);
            root(n, roots[i]);
            if (n.equals(r)) {
                out.println("Test " + (i + 1) + " passed: root(" + numbers[i]
                        + ", " + roots[i] + ") = " + results[i]);
            } else {
                out.println("*** Test " + (i + 1) + " failed: root("
                        + numbers[i] + ", " + roots[i] + ") expected <"
                        + results[i] + "> but was <" + n + ">");
            }
        }

        out.close();
    }

}
```