```java
import components.naturalnumber.NaturalNumber;

/**
 * CSE 2221 Project #6. Program with implementation of {@code NaturalNumber}
 * secondary operation {@code root} implemented as static method.
 *
 * @author Faye Leigh
 *
 */
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) {
        assert n != null : "Violation of: n is  not null";
        assert r >= 2 : "Violation of: r >= 2";

        final NaturalNumber one = new NaturalNumber2(1);
        final NaturalNumber two = new NaturalNumber2(2);
        NaturalNumber lowEnough = new NaturalNumber2(0);
        NaturalNumber tooHigh = new NaturalNumber2(n);
        boolean rootFound = false;

        /*
         * If n is 0 or 1, do nothing since the root of 0 is always 0, and the
         * root of 1 is always 1. If n is greater than 1, find the root
         */
        if (n.compareTo(one) > 0) {
            while (!rootFound) {
                /*
                 * Find new guess value g = (high - low) / 2 + low
                 */
                NaturalNumber g = new NaturalNumber2(tooHigh);
                g.subtract(lowEnough);
                g.divide(two);
                g.add(lowEnough);

                /*
                 * Find the guess raised to power r
                 */
                NaturalNumber gPow = new NaturalNumber2(g);
                gPow.power(r);

                /*
                 * If g^r > n, the root of r is less than g so set tooHigh to g.
                 * If g^r < n, the root of n is at least g so set lowEnough to g
                 */
                if (gPow.compareTo(n) > 0) {
                    tooHigh = g;
                } else {
                    lowEnough = g;
                }

                /*
                 * Exit loop if the difference between high and low bound is 1
                 */
                NaturalNumber tmp = new NaturalNumber2(tooHigh);
                tmp.subtract(lowEnough);
                if (tmp.compareTo(one) == 0) {
                    rootFound = true;
                }
            }
            n.copyFrom(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();
    }
}
```