```java
import components.naturalnumber.NaturalNumber;

/**
 * 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) {
        assert n != null : "Violation of: n is  not null";
        assert r >= 2 : "Violation of: r >= 2";

        SimpleWriter out = new SimpleWriter1L();

        // declare natural number variables
        NaturalNumber low = n.newInstance();
        NaturalNumber high = n.newInstance();
        NaturalNumber difference = n.newInstance();
        // intermediate natural number to hold values
        NaturalNumber inter = n.newInstance();
        // guess that will be changed each iteration
        NaturalNumber guess = n.newInstance();
        // use zero to compare
        NaturalNumber zero = new NaturalNumber2(0);
        // use one to compare
        NaturalNumber one = new NaturalNumber2(1);
        // use two to divide natural numbers by 2
        NaturalNumber two = new NaturalNumber2(2);

        // similar to lab where it is one higher than starting number
        high.add(n);
        high.add(one);
        difference.add(high);

        // take the guess in between the high and low ranges, and power inter
        guess.copyFrom(high);
        guess.divide(two);
        inter.copyFrom(guess);
        inter.power(r);

        // checks if the difference is 1 or 0 (where you can no longer interval half)
        while (difference.compareTo(one) != 0
                && difference.compareTo(zero) != 0) {
            if (n.compareTo(inter) < 0) {
                // change range
                high.copyFrom(guess);
                // get new guess value
                guess.divide(two);
                // compute difference
                difference.copyFrom(high);
                difference.subtract(low);

            } else {
                // change range
                low.copyFrom(guess);
                // compute difference
                difference.copyFrom(high);
                difference.subtract(low);
                // get new guess value
                inter.copyFrom(difference);
                inter.divide(two);
                guess.add(inter);

            }
            inter.copyFrom(guess);
            inter.power(r);

        }
        // lower inclusive use that value
        n.copyFrom(low);
    }

    /**
     * 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();
    }

}

```