```java
import components.naturalnumber.NaturalNumber;

/**
 * Program with implementation of {@code NaturalNumber} secondary operation
 * {@code root} implemented as static method.
 *
 * @author Isaac Frank
 *
 */
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";

        // initial bounds for the interval
        NaturalNumber lowEnough = new NaturalNumber2(0);
        n.increment();
        NaturalNumber tooHigh = new NaturalNumber2(n);
        n.decrement();

        // finding the average of lowEnough and tooHigh to halve the interval
        NaturalNumber two = new NaturalNumber2(2);
        NaturalNumber midInterval = new NaturalNumber2();
        midInterval.add(lowEnough);
        midInterval.add(tooHigh);
        midInterval.divide(two);
        NaturalNumber guess = power(midInterval, r);

        midInterval.increment();
        // iterating through until midInterval = the rth root of n
        while (guess.compareTo(n) > 0
                || power(midInterval, r).compareTo(n) <= 0) {
            midInterval.decrement();

            // halving the interval
            if (guess.compareTo(n) <= 0) {
                lowEnough.transferFrom(midInterval);
            } else {
                tooHigh.transferFrom(midInterval);
            }

            // resetting midInterval
            midInterval = new NaturalNumber2(tooHigh);
            midInterval.add(lowEnough);
            midInterval.divide(two);
            guess = power(midInterval, r);
            midInterval.increment();
        }

        // changing n to be the rth root of #n
        midInterval.decrement();
        n.transferFrom(midInterval);
    }

    /**
     * Returns a NaturalNumber, the power of {@code n} to the {@code p}-th
     * power.
     *
     * @param n
     *            the base of the power
     * @param p
     *            the exponent of the power
     * @return n raised to the p power
     * @requires p >= 0
     * @ensures n = #n and power = n ^ p
     */
    public static NaturalNumber power(NaturalNumber n, int p) {

        NaturalNumber ans = new NaturalNumber2(1);

        // recursive call if p > 1
        if (p > 1) {
            ans = power(n, p / 2);
            ans.multiply(power(n, p / 2));
        }

        if (p % 2 != 0) {
            ans.multiply(n);
        }
        return ans;
    }

    /**
     * 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();
    }
}
```