```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 Gabe Azzarita
 *
 */
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";

        // Constants needing for comparisons and division
        NaturalNumber one = new NaturalNumber2(1);
        NaturalNumber two = new NaturalNumber2(2);

        // Creating low (0) and high (n+1) for interval
        NaturalNumber low = new NaturalNumber2();
        NaturalNumber high = new NaturalNumber2(n);
        high.increment();

        NaturalNumber value = new NaturalNumber2();
        NaturalNumber difference = new NaturalNumber2(high);
        NaturalNumber guess = new NaturalNumber2(high);
        guess.divide(two);
        value.copyFrom(guess);

        // While (high - low) > 1
```

```
    while ((difference.compareTo(one) > 0)) {
        value.power(r);

        /**
         * If our value is smaller than n we set lower boundary to guess, if
         * it's bigger, we set our upper boundary to guess
         */
        if (value.compareTo(n) < 1) {
            low.copyFrom(guess);
        } else {
            high.copyFrom(guess);
        }

        // Change guess to be halfway point of new interval
        guess.clear();
        guess.add(high);
        guess.add(low);
        guess.divide(two);
        value.copyFrom(guess);

        // Update difference
        difference.copyFrom(high);
        difference.subtract(low);
    }
    n.copyFrom(guess);
}

/**
 * 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",
            "6189700196426901374495621111",
            "16225927682921336339157801028812 7",
            "17014118346046923173168730371588410 5727" };
    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();
    }

}
```