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

        /*
         * Constants to use during while loop
         */
        NaturalNumber one = new NaturalNumber2(1);
        NaturalNumber two = new NaturalNumber2(2);

```

```
42          /*
43           * Creates lowerBound and upperBound. upperBound is n+1
44           */
45          NaturalNumber lowerBound = new NaturalNumber2(0);
46          NaturalNumber upperBound = new NaturalNumber2(n);
47          upperBound.increment();
48
49          /*
50           * Creates guess variable
51           */
52          NaturalNumber guess = new NaturalNumber2();
53
54          /*
55           * This is a clone of guess variable so that functions
   that are done on
56           * it aren't copied over to the guess variable
57           */
58          NaturalNumber temp = new NaturalNumber2();
59
60          /*
61           * creates difference between both bounds
62           */
63          NaturalNumber difference = new NaturalNumber2(lowerBound);
64          difference.add(upperBound);
65
66          while (difference.compareTo(one) > 0) {
67              /*
68               * Halving part, guess will equal to the middle of
   upperBound and
69               * lowerBound
70               */
71              guess.copyFrom(lowerBound);
72              guess.add(upperBound);
73              guess.divide(two);
74
75              /*
76               * temp will then be copied from guess so that the
   power function
77               * doesn't affect guess
78               */
79              temp.copyFrom(guess);
80              temp.power(r);
81
82              /*
```

```
 83              * if temp^r is less than the original number, then
    the lowerBound
 84              * will equal the midpoint. Else the upperBound will
    equal the
 85              * midpoint, thereby eliminating half of the options
 86              */
 87             if (temp.compareTo(n) <= 0) {
 88                 lowerBound.copyFrom(guess);
 89             } else {
 90                 upperBound.copyFrom(guess);
 91             }
 92
 93             /*
 94              * The difference will then be recalculated to check
    if they are a
 95              * single unit apart. If they are, then the loop ends.
 96              */
 97             difference.copyFrom(upperBound);
 98             difference.subtract(lowerBound);
 99
100         }
101
102         /*
103          * the loop will end with lowerBound equaling the number
    we want to
104          * return, so n will become the lowerBound
105          */
106         n.copyFrom(lowerBound);
107     }
108
109     /**
110      * Main method.
111      *
112      * @param args
113      *            the command line arguments
114      */
115     public static void main(String[] args) {
116         SimpleWriter out = new SimpleWriter1L();
117
118         final String[] numbers = { "0", "1", "13", "1024",
    "189943527", "0",
119                 "1", "13", "4096", "189943527", "0", "1", "13",
    "1024",
120                 "189943527", "82", "82", "82", "82", "82", "9",
```

```
    "27", "81",
121                 "243", "143489073", "2147483647", "2147483648",
122                 "92233720368547758O7", "9223372036854775808",
123                 "6189700196426901374495621111",
124                 "16225927682921336339157801028812 7",
125                 "170141183460469231731687303715884105727" };
126         final int[] roots = { 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 15,
    15, 15, 15, 15,
127                 2, 3, 4, 5, 15, 2, 3, 4, 5, 15, 2, 2, 3, 3, 4, 5,
    6 };
128         final String[] results = { "0", "1", "3", "32", "13782",
    "0", "1", "2",
129                 "16", "574", "0", "1", "1", "1", "3", "9", "4",
    "3", "2", "1",
130                 "3", "3", "3", "3", "3", "46340", "46340",
    "2097151", "2097152",
131                 "4987896", "2767208", "2353973" };
132
133         for (int i = 0; i < numbers.length; i++) {
134             NaturalNumber n = new NaturalNumber2(numbers[i]);
135             NaturalNumber r = new NaturalNumber2(results[i]);
136             root(n, roots[i]);
137             if (n.equals(r)) {
138                 out.println("Test " + (i + 1) + " passed: root(" +
    numbers[i]
139                         + ", " + roots[i] + ") = " + results[i]);
140             } else {
141                 out.println("*** Test " + (i + 1) + " failed:
    root("
142                         + numbers[i] + ", " + roots[i] + ")
    expected <"
143                         + results[i] + "> but was <" + n + ">");
144             }
145         }
146
147         out.close();
148     }
149
150 }
```