```java
import components.simplereader.SimpleReader;

/**
 * Put a short phrase describing the program here.
 *
 * @author David Park
 *
 */
public final class ABCDGuesser1 {

    /**
     * No argument constructor--private to prevent instantiation.
     */
    private ABCDGuesser1() {
    }

    /**
     * Repeatedly asks the user for a positive real number until the user enters
     * one. Returns the positive real number.
     *
     * @param in
     *            the input stream
     * @param out
     *            the output stream
     * @return a positive real number entered by the user
     */
    private static double getPositiveDouble(SimpleReader in, SimpleWriter out) {
        /*
         * Asks user to input a a positive real double. Then parse the double
         * from input string and return it.
         */
        double mu = 0;
        String s = "";
        while (mu <= 0) {
            out.println("Please input a positive double: ");
            s = in.nextLine();
            if (FormatChecker.canParseDouble(s)) {
                // this checks to see if the number is valid to be parsed into a double
                mu = Double.parseDouble(s);
                if (mu <= 0) {
                    out.println("The number must be positive. enter again");
                }
            } else {
                out.println("not a valid double. enter again");
            }
        }
        return mu;
    }

    /**
     * Repeatedly asks the user for a positive real number not equal to 1.0
     * until the user enters one. Returns the positive real number.
     *
     * @param in
     *            the input stream
     * @param out
     *            the output stream
```

```
 62      * @return a positive real number not equal to 1.0 entered by the user
 63      */
 64     private static double getPositiveDoubleNotOne(SimpleReader in,
 65             SimpleWriter out) {
 66         /*
 67          * Asks user to input a positive real double that is NOT equal to 1.0.
 68          * Then parse the double from input string and return it.
 69          */
 70         double input = 0;
 71         String s = "";
 72         while ((input <= 0 && input != 1) || !FormatChecker.canParseDouble(s)) {
 73             //repeating while input is less than or equal to
 74             out.println("Please input a positive double: ");
 75             s = in.nextLine();
 76             if (FormatChecker.canParseDouble(s)
 77                     // check if the double inside the string can be parsed.
 78                     && Double.parseDouble(s) != 1.0) {
 79                 input = Double.parseDouble(s);
 80                 if (input <= 0) {
 81                     out.println("The number must be positive. enter again");
 82                 }
 83             } else {
 84                 out.println("not a valid double. enter again");
 85             }
 86         }
 87         return input;
 88     }
 89
 90     /**
 91      * Main method.
 92      *
 93      * @param args
 94      *            the command line arguments
 95      */
 96     public static void main(String[] args) {
 97         SimpleReader in = new SimpleReader1L();
 98         SimpleWriter out = new SimpleWriter1L();
 99
100         double mu = getPositiveDouble(in, out);
101
102         // Prompt user for four input values, ensuring they're positive and not one
103         double input1 = getPositiveDoubleNotOne(in, out);
104         double input2 = getPositiveDoubleNotOne(in, out);
105         double input3 = getPositiveDoubleNotOne(in, out);
106         double input4 = getPositiveDoubleNotOne(in, out);
107
108         // Define an array of exponent values to try in the approximation
109         final double[] exponents = { -5.0, -4.0, -3.0, -2.0, -1.0, -1.0 / 2.0,
110                 -1.0 / 3.0, -1.0 / 4.0, 0.0, 1.0 / 4.0, 1.0 / 3.0, 1.0 / 2.0,
111                 1.0, 2.0, 3.0, 4.0, 5.0 };
112
113         // Initialize variables to hold the best approx after the de jager
114         // approximation found and the corresponding exponents
115         double bestApprox = 0;
116         double bestA = 0, bestB = 0, bestC = 0, bestD = 0;
117         final double hundred = 100;
118         // the 4 while loops will be calculating the de jager value.
```

```
119         int i = 0;
120         while (i < exponents.length) { // First loop selects an exponent for a
121             double a = exponents[i];
122             int j = 0;
123             while (j < exponents.length) { // Second loop selects an exponent for b
124                 double b = exponents[j];
125                 int k = 0;
126                 while (k < exponents.length) { // Third loop selects an exponent for c
127                     double c = exponents[k];
128                     int l = 0;
129                     // Fourth loop selects an exponent for d
130                     while (l < exponents.length) {
131                         double d = exponents[l];
132                         // Update the best approximation and
133                         // exponents if the current one is closer to mu
134                         double currentApproximation = Math.pow(input1, a)
135                                 // Calculate the current approximation
136                                 // using the selected exponents
137                                 * Math.pow(input2, b) * Math.pow(input3, c)
138                                 * Math.pow(input4, d);
139                         if (Math.abs(mu - currentApproximation) < Math
140                                 .abs(mu - bestApprox)) {
141                             bestApprox = currentApproximation;
142                             bestA = a;
143                             bestB = b;
144                             bestC = c;
145                             bestD = d;
146                         }
147                         l++;
148                     }
149                     k++;
150                 }
151                 j++;
152             }
153             i++;
154         }
155 
156         // Output the best approximation found and its details
157 
158         out.println("Best approximation: " + bestApprox);
159         out.println("Exponents: a=" + bestA + ", b=" + bestB + ", c=" + bestC
160                 + ", d=" + bestD);
161         out.println("Smallest relative error: ");
162         out.print((bestApprox - mu) / mu * hundred, 2, false);
163         out.print("%");
164 
165         /*
166          * Close input and output streams
167          */
168         in.close();
169         out.close();
170     }
171 
172 }
173 
```