```java
import components.simplereader.SimpleReader;

/**
 * Put a short phrase describing the program here.
 *
 * @author David Park
 *
 */
public final class ABCDGuesser2 {

    /**
     * No argument constructor--private to prevent instantiation.
     */
    private ABCDGuesser2() {
    }

    /**
     * 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 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
```

```java
     * @return a positive real number not equal to 1.0 entered by the user
     */
    private static double getPositiveDoubleNotOne(SimpleReader in,
            SimpleWriter out) {
        /*
         * Asks user to input a a positive real double that is NOT equal to 1.0.
         * Then parse the double from input string and return it.
         */
        double input = 0;
        String s = "";
        while ((input <= 0 && input != 1) || !FormatChecker.canParseDouble(s)) {
            //repeating while input is less than or equal to
            out.println("Please input a positive double: ");
            s = in.nextLine();
            if (FormatChecker.canParseDouble(s)
                    // check if the double inside the string can be parsed.
                    && Double.parseDouble(s) != 1.0) {
                input = Double.parseDouble(s);
                if (input <= 0) {
                    out.println("The number must be positive. enter again");
                }
            } else {
                out.println("not a valid double. enter again");
            }
        }
        return input;
    }

    /**
     * Searches for the best approximation of a given constant (mu) using the de
     * Jager formula with four input values and a set of exponents. It iterates
     * through all combinations of the exponents for the input values to find
     * the combination that results in the value closest to mu, minimizing the
     * relative error.
     *
     *
     * @param out
     *            output stream
     * @param constantMu
     *            user entered number to be approximated
     * @param exponents
     *            final double array where the exponents will be used
     * @param input1
     *            first user entered number to be approximated by raising to
     *            different powers
     * @param input2
     *            second user entered number to be approximated by raising to
     *            different powers
     * @param input3
     *            third user entered number to be approximated by raising to
     *            different powers
     * @param input4
     *            fourth user entered number to be approximated by raising to
     *            different powers
     */
    private static void bestApproximationSearch(SimpleWriter out,
            double constantMu, double input1, double input2, double input3,
```

```
119             double input4, double[] exponents) {
120         // Initialize variables to hold the best approx after the de jager
121         // approximation found and the corresponding exponents
122         double bestApproximation = 0;
123         double bestA = 0, bestB = 0, bestC = 0, bestD = 0;
124         final double hundred = 100;
125         // 4 four loops to solve for the de jager value
126         // First loop selects an exponent for i
127         for (int i = 0; i < exponents.length; i++) {
128             // Second loop selects an exponent for j
129             for (int j = 0; j < exponents.length; j++) {
130                 // Third loop selects an exponent for k
131                 for (int k = 0; k < exponents.length; k++) {
132                     // Fourth loop selects an exponent for l
133                     for (int l = 0; l < exponents.length; l++) {
134 
135                         // Update the best approximation and
136                         // exponents if the current one is closer to mu
137                         double currentApproximation = Math.pow(input1,
138                                 exponents[i]) * Math.pow(input2, exponents[j])
139                                 * Math.pow(input3, exponents[k])
140                                 * Math.pow(input4, exponents[l]);
141                         // Calculate the current approximation
142                         // using the selected exponents
143                         if (Math.abs(constantMu - currentApproximation) < Math
144                                 .abs(constantMu - bestApproximation)) {
145                             bestApproximation = currentApproximation;
146                             bestA = exponents[i];
147                             bestB = exponents[j];
148                             bestC = exponents[k];
149                             bestD = exponents[l];
150                         }
151                     }
152                 }
153             }
154         }
155 
156         // Output the best approximation found and its details
157         out.println("Best approximation: " + bestApproximation);
158         out.println("Exponents: a=" + bestA + ", b=" + bestB + ", c=" + bestC
159                 + ", d=" + bestD);
160         out.println("Smallest relative error: ");
161         out.print((bestApproximation - constantMu) / constantMu * hundred, 2,
162                 false);
163         out.print("%");
164 
165     }
166 
167     /**
168      * Main method.
169      *
170      * @param args
171      *            the command line arguments
172      */
173     public static void main(String[] args) {
174         SimpleReader in = new SimpleReader1L();
175         SimpleWriter out = new SimpleWriter1L();
```

```
176
177         double mu = getPositiveDouble(in, out);
178
179         // Prompt the user for four positive double values that are not equal to one.
180         // These values will be used in the approximation process.
181         double input1 = getPositiveDoubleNotOne(in, out);
182         double input2 = getPositiveDoubleNotOne(in, out);
183         double input3 = getPositiveDoubleNotOne(in, out);
184         double input4 = getPositiveDoubleNotOne(in, out);
185
186         // Define an array of exponent values to try in the approximation
187         final double[] exponents = { -5.0, -4.0, -3.0, -2.0, -1.0, -1.0 / 2.0,
188                 -1.0 / 3.0, -1.0 / 4.0, 0.0, 1.0 / 4.0, 1.0 / 3.0, 1.0 / 2.0,
189                 1.0, 2.0, 3.0, 4.0, 5.0 };
190
191         // Call the bestApproximationSearch method with the gathered inputs and exponents.
192         // This method will compute and print the best approximation to the value 'mu'
193         // using a combination of the input values raised to the power of the exponents.
194         bestApproximationSearch(out, mu, input1, input2, input3, input4,
195                 exponents);
196
197         /*
198          * Close input and output streams
199          */
200         in.close();
201         out.close();
202     }
203
204 }
205
```