

```
1 import components.simplereader.SimpleReader;
6
7 /**
8  * Guesses best values for a, b, c, d for de Jager's algorithm.
9  *
10 * @author Isaac Frank
11 *
12 */
13 public final class ABCDGuesser2 {
14
15     /**
16      * Private constructor so this utility class cannot be
17      instantiated.
18      */
19     private ABCDGuesser2() {
20         //not called
21     }
22
23     /**
24      * Repeatedly asks the user for a positive real number until
25      the user enters
26      * one. Returns the positive real number.
27      *
28      * @param in
29      *         the input stream
30      * @param out
31      *         the output stream
32      * @return a positive real number entered by the user
33      */
34     private static double getPositiveDouble(SimpleReader in,
35     SimpleWriter out) {
36         double num = -1;
37
38         // Asks user for input until a positive real number is
39         entered
40         while (num <= 0) {
41             out.print("Enter a positive real number: ");
42             String input = in.nextLine();
43             if (FormatChecker.canParseDouble(input)) {
44                 num = Double.parseDouble(input);
45                 if (num <= 0) {
46                     out.println("Number must be positive!");
47                 }
48             } else {
```

```
45         out.println("Must be a real number!");
46     }
47 }
48     return num;
49 }
50
51 /**
52  * Repeatedly asks the user for a positive real number not
53  * equal to 1.0
54  * until the user enters one. Returns the positive real
55  * number.
56  *
57  * @param in
58  *         the input stream
59  * @param out
60  *         the output stream
61  * @return a positive real number not equal to 1.0 entered by
62  *         the user
63  */
64 private static double getPositiveDoubleNotOne(SimpleReader in,
65         SimpleWriter out) {
66     double num = -1;
67
68     // Asks user for input until a positive real number != 1
69     // is entered
70     while (num <= 0 || num == 1.0) {
71         out.print("Enter a positive real number not equal to
72         1: ");
73         String input = in.nextLine();
74         if (FormatChecker.canParseDouble(input)) {
75             num = Double.parseDouble(input);
76             if (num <= 0 || num == 1) {
77                 out.println("Number cannot be 1, and must be
78                 positive!");
79             }
80         } else {
81             out.println("Must be a real number!");
82         }
83     }
84     return num;
85 }
86
87 /**
88  * Calculates the relative error for the current estimate of
```

```

    mu.
83     *
84     * @param percentConv
85     *         the double for the percent that makes up a whole
86     * @param estimate
87     *         current estimate to calculate relative error of
88     * @param mu
89     *         constant that algorithm is attempting to
estimate
90     * @return the relative error, calculated by  $100\% * |e - u| /$ 
u
91     */
92     private static double getRelError(double percentConv, double
estimate,
93         double mu) {
94         double relError = percentConv * Math.abs(estimate - mu) /
mu;
95         return relError;
96     }
97
98     /**
99     * Main method, uses getPositiveDouble and
getPositiveDoubleNotOne to accept
100    * user input, then applies the de Jager formula, calling
getRelError to
101    * calculate the relative error with each iteration.
102    *
103    * @param args
104    *         Java command line arguments
105    */
106    public static void main(String[] args) {
107        // Opening input and output streams
108        SimpleReader in = new SimpleReader1L();
109        SimpleWriter out = new SimpleWriter1L();
110
111        // Array of 17 numbers asserted by the Charming Theory
112        final double[] charmingNums = { -5, -4, -3, -2, -1, -.5,
-1.0 / 3, -.25,
113            0, .25, 1.0 / 3, .5, 1, 2, 3, 4, 5 };
114
115        // Getting input from user
116        double mu = getPositiveDouble(in, out);
117        double w = getPositiveDoubleNotOne(in, out);
118        double x = getPositiveDoubleNotOne(in, out);

```

```
119     double y = getPositiveDoubleNotOne(in, out);
120     double z = getPositiveDoubleNotOne(in, out);
121
122     // Declaring variables to be calculated and used in loops
123     double estimate = -1;
124     double leastEstimate = -1;
125     double relError = -1;
126     double minRelError = Double.MAX_VALUE;
127     final double percentConv = 100;
128
129     /*
130     * Iterating through all combinations of w^a*x^b*y^c*z^d,
    then storing
131     * the least relative error and the estimate with the
    least relative
132     * error
133     */
134     for (int i = 0; i < charmingNums.length; i++) {
135         for (int j = 0; j < charmingNums.length; j++) {
136             for (int k = 0; k < charmingNums.length; k++) {
137                 for (int l = 0; l < charmingNums.length; l++)
138                     {
139                         estimate = Math.pow(w, charmingNums[i])
140                             * Math.pow(x, charmingNums[j])
141                             * Math.pow(y, charmingNums[k])
142                             * Math.pow(z, charmingNums[l]);
143                         relError = getRelError(percentConv,
144                             estimate, mu);
145                         if (relError < minRelError) {
146                             minRelError = relError;
147                             leastEstimate = estimate;
148                         }
149                     }
150             }
151         }
152     }
153
154     // Printing the closest estimate and rounded relative
    error of estimate
155     out.println("Estimate: " + leastEstimate);
156     out.print("Relative Error: ");
157     out.print(minRelError, 2, false);
158     out.println("%");
```

```
158         // Closing input and output streams
159         in.close();
160         out.close();
161     }
162
163 }
164
```