

```
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 ABCDGuesser1 {
14
15     /**
16      * Private constructor so this utility class cannot be
17      instantiated.
18      */
19     private ABCDGuesser1() {
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  * Main method, uses getPositiveDouble and
```

```

    getPositiveDoubleNotOne to accept
83     * user input, then applies the de Jager formula.
84     *
85     * @param args
86     *         Java command line arguments
87     */
88     public static void main(String[] args) {
89         // Opening input and output streams
90         SimpleReader in = new SimpleReader1L();
91         SimpleWriter out = new SimpleWriter1L();
92
93         // Array of 17 numbers asserted by the Charming Theory
94         final double[] charmingNums = { -5, -4, -3, -2, -1, -.5,
-1.0 / 3, -.25,
95             0, .25, 1.0 / 3, .5, 1, 2, 3, 4, 5 };
96
97         // Getting input from user
98         double mu = getPositiveDouble(in, out);
99         double w = getPositiveDoubleNotOne(in, out);
100        double x = getPositiveDoubleNotOne(in, out);
101        double y = getPositiveDoubleNotOne(in, out);
102        double z = getPositiveDoubleNotOne(in, out);
103
104        // Declaring variables to be calculated and used in loops
105        double estimate = -1;
106        double leastEstimate = -1;
107        double relError = -1;
108        double minRelError = Double.MAX_VALUE;
109        final double percentConv = 100;
110
111        int i = 0;
112        int j = 0;
113        int k = 0;
114        int l = 0;
115
116        /*
117        * Iterating through all combinations of w^a*x^b*y^c*z^d,
then storing
118        * the least relative error and the estimate with the
least relative
119        * error
120        */
121        while (i < charmingNums.length) {
122            j = 0;

```

```
123         while (j < charmingNums.length) {
124             k = 0;
125             while (k < charmingNums.length) {
126                 l = 0;
127                 while (l < charmingNums.length) {
128                     estimate = Math.pow(w, charmingNums[i])
129                         * Math.pow(x, charmingNums[j])
130                         * Math.pow(y, charmingNums[k])
131                         * Math.pow(z, charmingNums[l]);
132                     relError = percentConv * Math.abs(estimate
133 - mu) / mu;
134                     if (relError < minRelError) {
135                         minRelError = relError;
136                         leastEstimate = estimate;
137                     }
138                     l++;
139                 }
140                 k++;
141             }
142             j++;
143         }
144         i++;
145     }
146
147     // Printing the closest estimate and rounded relative
148     error of estimate
149     out.println("Estimate: " + leastEstimate);
150     out.print("Relative Error: ");
151     out.print(minRelError, 2, false);
152     out.println("%");
153
154     // Closing input and output streams
155     in.close();
156     out.close();
157 }
158
```