```
1 import components.simplereader.SimpleReader;
7 /**
8 * Guesses best values for a, b, c, d for de Jager's algorithm.
10 * @author Isaac Frank
11 *
12 */
13 public final class ABCDGuesser1 {
14
15
      /**
16
       * Private constructor so this utility class cannot be
  instantiated.
17
       */
      private ABCDGuesser1() {
18
19
         // Not called
20
21
22
23
       * Repeatedly asks the user for a positive real number until
  the user enters
24
       * one. Returns the positive real number.
25
26
       * @param in
27
                    the input stream
28
       * @param out
29
                    the output stream
30
       * @return a positive real number entered by the user
31
32
      private static double getPositiveDouble(SimpleReader in,
  SimpleWriter out)
33
          double num = -1:
34
35
          // Asks user for input until a positive real number is
  entered
36
          while (num <= 0
37
              out.print("Enter a positive real number: ");
38
              String input = in nextLine(
              if (FormatChecker_canParseDouble(input)) {
39
40
                  num = Double.parseDouble(input);
41
                  if (num <= 0
                      out.println("Number must be positive!");
42
43
44
              } 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
  equal to 1.0
53
       * until the user enters one. Returns the positive real
  number.
54
       *
55
       * @param in
56
                    the input stream
57
       * @param out
58
                    the output stream
59
       * @return a positive real number not equal to 1.0 entered by
  the user
60
       */
61
      private static double getPositiveDoubleNotOne(SimpleReader in,
62
              SimpleWriter out
          double num = -1:
63
64
65
          // Asks user for input until a positive real number != 1
  is entered
          while (num <= 0 || num == 1.0) {
66
              out.print("Enter a positive real number not eqaul to
67
68
              String input = in nextLine();
              if (FormatChecker_canParseDouble(input)) {
69
70
                  num = Double.parseDouble(input);
71
                  if (num <= 0 | num == 1)
                      out println ("Number cannot be 1, and must be
72
  positive!");
73
74
              } else {
75
                  out.println("Must be a real number!");
76
77
78
         return num;
79
80
81
82
       * 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 \begin{bmatrix} -5, -4, -3, -2, -1, -.5 \end{bmatrix}
    1.0 / 3, -.25,
                    0, .25, 1.0 / 3, .5, 1, 2, 3, 4, 5 };
 95
 96
 97
            // Getting input from user
 98
            double mu = getPositiveDouble(in, out);
99
            double w = getPositiveDoubleNotOne(in, out);
            double x = getPositiveDoubleNotOne(in, out);
100
101
            double v = getPositiveDoubleNotOne(in, out);
102
            double z = getPositiveDoubleNotOne(in, out);
103
104
           // Declaring variables to be calculated and used in loops
105
            double estimate = -1:
            double leastEstimate = -1:
106
107
            double relError = -1:
108
            double minRelError = Double.MAX VALUE;
109
            final double percentConv = 100;
110
111
            int i = 0:
112
            int i = 0;
113
            int k = 0;
114
           int l = 0:
115
116
            * Iterating through all combinations of w^a*x^b*y^c*z^d,
117
   then storing
             * the least relative error and the estimate with the
118
   least relative
119
            * error
120
            */
121
           while (i < charmingNums_length) {</pre>
122
                i = 0;
```

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