```java
import components.simplereader.SimpleReader;

/**
 * Guesses best values for a, b, c, d for de Jager's algorithm.
 *
 * @author Isaac Frank
 *
 */
public final class ABCDGuesser2 {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private ABCDGuesser2() {
        //not called
    }

    /**
     * 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) {
        double num = -1;

        // Asks user for input until a positive real number is entered
        while (num <= 0) {
            out.print("Enter a positive real number: ");
            String input = in.nextLine();
            if (FormatChecker.canParseDouble(input)) {
                num = Double.parseDouble(input);
                if (num <= 0) {
                    out.println("Number must be positive!");
                }
            } else {
                out.println("Must be a real number!");
            }
        }
        return num;
    }

    /**
     * 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
     * @return a positive real number not equal to 1.0 entered by the user
     */
    private static double getPositiveDoubleNotOne(SimpleReader in,
            SimpleWriter out) {
        double num = -1;

        // Asks user for input until a positive real number != 1 is entered
        while (num <= 0 || num == 1.0) {
            out.print("Enter a positive real number not eqaul to 1: ");
            String input = in.nextLine();
            if (FormatChecker.canParseDouble(input)) {
                num = Double.parseDouble(input);
                if (num <= 0 || num == 1) {
                    out.println("Number cannot be 1, and must be positive!");
                }
            } else {
                out.println("Must be a real number!");
            }
        }
        return num;
    }

    /**
     * Calculates the relative error for the current estimate of
```

```java
mu.
     *
     * @param percentConv
     *            the double for the percent that makes up a whole
     * @param estimate
     *            current estimate to calculate relative error of
     * @param mu
     *            constant that algorithm is attempting to estimate
     * @return the relative error, calculated by 100% * |e - u| / u
     */
    private static double getRelError(double percentConv, double estimate,
            double mu) {
        double relError = percentConv * Math.abs(estimate - mu) / mu;
        return relError;
    }

    /**
     * Main method, uses getPositiveDouble and getPositiveDoubleNotOne to accept
     * user input, then applies the de Jager formula, calling getRelError to
     * calculate the relative error with each iteration.
     *
     * @param args
     *            Java command line arguments
     */
    public static void main(String[] args) {
        // Opening input and output streams
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();

        // Array of 17 numbers asserted by the Charming Theory
        final double[] charmingNums = { -5, -4, -3, -2, -1, -.5, -1.0 / 3, -.25,
                0, .25, 1.0 / 3, .5, 1, 2, 3, 4, 5 };

        // Getting input from user
        double mu = getPositiveDouble(in, out);
        double w = getPositiveDoubleNotOne(in, out);
        double x = getPositiveDoubleNotOne(in, out);
        double y = getPositiveDoubleNotOne(in, out);
        double z = getPositiveDoubleNotOne(in, out);

        // Declaring variables to be calculated and used in loops
        double estimate = -1;
        double leastEstimate = -1;
        double relError = -1;
        double minRelError = Double.MAX_VALUE;
        final double percentConv = 100;

        /*
         * Iterating through all combinations of w^a*x^b*y^c*z^d, then storing
         * the least relative error and the estimate with the least relative
         * error
         */
        for (int i = 0; i < charmingNums.length; i++) {
            for (int j = 0; j < charmingNums.length; j++) {
                for (int k = 0; k < charmingNums.length; k++) {
                    for (int l = 0; l < charmingNums.length; l++) {
                        estimate = Math.pow(w, charmingNums[i])
                                * Math.pow(x, charmingNums[j])
                                * Math.pow(y, charmingNums[k])
                                * Math.pow(z, charmingNums[l]);
                        relError = getRelError(percentConv, estimate, mu);
                        if (relError < minRelError) {
                            minRelError = relError;
                            leastEstimate = estimate;
                        }
                    }
                }
            }
        }

        // Printing the closest estimate and rounded relative error of estimate
        out.println("Estimate: " + leastEstimate);
        out.print("Relative Error: ");
        out.print(minRelError, 2, false);
        out.println("%");

        // Closing input and output streams
        in.close();
        out.close();
    }

}
```