```java
import components.simplereader.SimpleReader;

/**
 * CSE 2221 Project #3.
 *
 * @author Faye Leigh
 */
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) {
        double output = 0.0;
        boolean flag = true;
        String input;

        /*
         * Run until user input satisfies conditions
         */
        while (flag) {
            out.print("Please enter a positive number: ");
            input = in.nextLine();

            /*
             * Checks that input contains a number and is positive
             */
            if (FormatChecker.canParseDouble(input)) {
                output = Double.parseDouble(input);
                if (output > 0) {
                    flag = false;
                } else {
                    out.println("Number was not positive.");
                }
            } else {
                out.println("Input was not an number.");
            }
        }
        return output;
    }

    /**
     * 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 output = 0.0;
        boolean flag = true;
        String input;

        /*
         * Run until user input satisfies conditions
         */
        while (flag) {
            out.print("Please enter a number greater than 1.0: ");
            input = in.nextLine();

            /*
             * Checks that input contains a number and is greater than 1.0
             */
            if (FormatChecker.canParseDouble(input)) {
                output = Double.parseDouble(input);
                if (output > 1.0) {
                    flag = false;
                } else {
                    out.println("Number was not greater than 1.0");
                }
            } else {
                out.println("Input was not an number.");
            }
        }
        return output;
    }

    /**
     * Runs the de Jager approximation. Returns a real number.
     *
     * @param w
     *            first personal number
     * @param i
     *            first exponent
     * @param x
     *            second personal number
     * @param j
     *            second exponent
     * @param y
     *            third personal number
     * @param k
     *            third exponent
     * @param z
     *            fourth personal number
     * @param l
     *            fourth exponent
     * @return the real number approximation
     */
    private static double getApproximate(double w, double i, double x, double j,
            double y, double k, double z, double l) {
        return Math.pow(w, i) * Math.pow(x, j) * Math.pow(y, k)
                * Math.pow(z, l);
    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments
```

```java
     */
    public static void main(String[] args) {
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();
        final double[] deJagerNum = { -5.0, -4.0, -3.0, -2.0, -1.0, -0.5,
                -1.0 / 3.0, -0.25, 0, 0.25, 1.0 / 3.0, 0.5, 1.0, 2.0, 3.0, 4.0,
                5.0 };
        final double toPercent = 100;
        final int size = deJagerNum.length;
        double a = 0, b = 0, c = 0, d = 0, w = 0, x = 0, y = 0, z = 0, mu = 0,
                approximate = 0, bestApproximate = 0, eps = 1.0;
        int i = 0, j = 0, k = 0, l = 0;

        /*
         * Asks user for a positive number and 4 more numbers greater than 1
         */
        out.println(
                "Choose a physical or mathematical constant you wish to approximate.");
        mu = getPositiveDouble(in, out);
        out.println(
                "Enter 4 numbers greater than 1.0 that have some personal meaning.");
        out.println("First number (w)");
        w = getPositiveDoubleNotOne(in, out);
        out.println("Second number (x)");
        x = getPositiveDoubleNotOne(in, out);
        out.println("Third number (y)");
        y = getPositiveDoubleNotOne(in, out);
        out.println("Fourth number (z)");
        z = getPositiveDoubleNotOne(in, out);

        for (i = 0; i < size; i++) {
            for (j = 0; j < size; j++) {
                for (k = 0; k < size; k++) {
                    for (l = 0; l < size; l++) {
                        /*
                         * Approximates mu with all possible combinations of the
                         * 17 de Jager exponents
                         */
                        approximate = getApproximate(w, deJagerNum[i], x,
                                deJagerNum[j], y, deJagerNum[k], z,
                                deJagerNum[l]);
                        /*
                         * Tests for lowest relative error for each approximate.
                         * Saves approximate, error, and exponents if lowest
                         * error is found
                         */
                        if (Math.abs(approximate - mu) / mu < eps) {
                            eps = Math.abs(approximate - mu) / mu;
                            bestApproximate = approximate;
                            a = deJagerNum[i];
                            b = deJagerNum[j];
                            c = deJagerNum[k];
                            d = deJagerNum[l];
                        }
                    }
                }
            }
        }

        out.println();
        out.println("Constant: " + mu);
        out.println("Best approximate: " + bestApproximate);
        out.print("Relative error: ");
        out.print(eps * toPercent, 2, false);
        out.println("%");
        out.println("Formula: (w^a)(x^b)(y^c)(z^d)");
        out.println("w: " + w + ", x: " + x + ", y: " + y + ", z: " + z);
        out.println("a: " + a + ", b: " + b + ", c: " + c + ", d: " + d);

        in.close();
        out.close();
    }

}

```