```java
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;
import components.utilities.FormatChecker;

/**
 * Program that users de Jager formula to estimate a number.
 *
 * @author Gabe Azzarita
 *
 */
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 positiveDouble = -1;
        while (positiveDouble < 0) {
            out.print("Enter a positive real number to approximate: ");
            String input = in.nextLine();
            if (FormatChecker.canParseDouble(input)) {
                positiveDouble = Double.parseDouble(input);
            }
        }
        return positiveDouble;
    }

    /**
     * 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 positiveDouble = -1;
        while (positiveDouble <= 1.0) {
            out.print("Enter a real number > 1: ");
            String input = in.nextLine();
            if (FormatChecker.canParseDouble(input)) {
                positiveDouble = Double.parseDouble(input);
            }
        }
        return positiveDouble;
    }

    /**
     * Calculates the percent error between the target number and the best
     * estimate.
     *
     * @param bestEst
     *            best approximation calculated by the program
     * @param mu
     *            the target number
     * @return the relative error between the approximation and actual
     */
    private static double getRelativeError(double bestEst, double mu) {
        final double relError = 100.0 * (Math.abs(bestEst - mu) / mu);
        return relError;
    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments
     */
    public static void main(String[] args) {
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();

        // Get desired numbers from user
        double mu = getPositiveDouble(in, out);
        double numOne = getPositiveDoubleNotOne(in, out);
        double numTwo = getPositiveDoubleNotOne(in, out);
        double numThree = getPositiveDoubleNotOne(in, out);
        double numFour = getPositiveDoubleNotOne(in, out);

        final double[] exponentArray = { -5, -4, -3, -2, -1, -(1.0 / 2.0),
                -(1.0 / 3.0), -(1.0 / 4.0), 0, (1.0) / (4.0), (1.0 / 3.0),
                (1.0 / 2.0), 1, 2, 3, 4, 5 };
```

```
        // Integers for our current estimation and our best estimation
        double tempEst = 0.0;
        double bestEst = 0.0;
        int bestH = 0;
        int bestI = 0;
        int bestJ = 0;
        int bestK = 0;

        int h = 0, i = 0, j = 0, k = 0;

        // Nested loop that tests all combinations of exponents
        for (h = 0; h < exponentArray.length - 1; h++) {
            i = 0;
            for (i = 0; i < exponentArray.length - 1; i++) {
                j = 0;
                for (j = 0; j < exponentArray.length - 1; j++) {
                    k = 0;
                    for (k = 0; k < exponentArray.length - 1; k++) {
                        tempEst = ((Math.pow(numOne, exponentArray[h]))
                                * (Math.pow(numTwo, exponentArray[i]))
                                * (Math.pow(numThree, exponentArray[j]))
                                * (Math.pow(numFour, exponentArray[k])));
                        // Seeing if current estimate is better than best estimate
                        if ((Math.abs(mu - tempEst)) < (Math
                                .abs(mu - bestEst))) {
                            bestEst = tempEst;
                            bestH = h;
                            bestI = i;
                            bestJ = j;
                            bestK = k;
                        }
                    }
                }
            }
        }
        // Final statements
        out.println("h: " + exponentArray[bestH] + " i: " + exponentArray[bestI]
                + " j: " + exponentArray[bestJ] + " k: "
                + exponentArray[bestK]);
        out.println("Best estimate: " + bestEst);
        out.print("Relative erorr: " + getRelativeError(bestEst, mu));

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

}
```