```
/**

================================================================================
 |   Source code:  ComputeE.java
 |        Author:  Alp Karavil
 |    Student ID:  5827197
 |    Assignment:  Program #1 - ComputeE
 |
 |            Course:  COP 3337 (Programming II)
 |           Section:  U00
 |        Instructor:  William Feild
 |        Due Date:   6 September 2018, by
         the beginning of class
 |
 |      I hereby certify that this
        collective work is my own
 |      and none of it is the work of any
        other person or entity.
 |      Alp Karavil
 |
 |  Language:  Java
 |  Compile/Run:
 |      javac ComputeE.java
 |      java ComputeE
 |
 |  +-------------------------------------------------------------------------
----
 |
 |  Description:  This program computes e to
    the decimal precision specified
    by the user.
 |
 |                This is possible through
          simulating the (1/n!) summation
          that starts at 0 and ends at
          infinity, until the difference
          between two iterations is less
          than the max error allowed,
          which depends on the precision
          specified by the task, which
          is 16 decimals in this case.
 |
 |        Input:  There is no input.
 |
 |       Output:  The program will print out the computed output, the
expected
         output of e (to the specified decimal precision) and the
         number of iterations (of the summation 1/n!) it took to get
         there.
 |
 |      Process:  The program's steps are as follows:
 |
 |                      1.  computeE() method is called
```

```
 |                      2.  max error allowed is calculated
 |                      3.  summation is simulated until delta (difference)
is
                       less than max error allowed
 |                      4.  Output is displayed to the user
 |
 |               No particular algorithms are used.  The Java BigDecimal
                 import is used to manipulate the summation total without
                 decimal underflow.
 |
 |   Required Features Not Included:  All required features are included.
 |
 |   Known Bugs:  None; the program operates correctly.
 |

*=========================================================================*
/

import java.math.BigDecimal;

/**
 * This class has 3 private methods. computeE(int decimalAccuracy) will
utilize
 * the two other methods (printOutput() and getMaxError() to print out a
 * formatted output including the computed value of e (to the precision
 * decided on), the expected value of e constant at desired decimal
precision,
 * and the number of iterations of the summation 1/n! from 0 to infinity

 * required to achieve the computed e value.
 */
public class ComputeE {

   // Constant E, (http://www-history.mcs.st-
and.ac.uk/HistTopics/e_10000.html)
   // java.math constant not used as it is rounded to 15 decimals.
   private static final BigDecimal E_38_DECIMALS =
           new BigDecimal("2.71828182845904523536028747135266249775");

   public static void main(String[] args)
   {
      computeE(16);
   }

   /**
    * Private method that calculates the natural number, e, to requested
decimal
    * accuracy. This calculation to  is achieved through simulating the
    * summation (1/n!) that starts at 0 and ends at infinity, which will be
    * stopped after reaching required decimal precision. After the
    * calculation, it will be printed out in such format:
    *
    * Computed value of e: --------
```

```
     * Expected value of e: 2.7182818284590452
     * Required itarations: ---- (Number of times (1/n!) summation was
     * executed to get the result.
     *
     * @param decimalPrecision
     */
    private static void computeE(int decimalPrecision)
    {
        //Final integer value for requested decimal accuracy
        final int decimalAccuracy = decimalPrecision;

        // Holds the summation value of the current iteration of 1/n!
        BigDecimal totalSum = new BigDecimal(0.0);
        // Holds the previous totalSum value for delta calculation.
        BigDecimal lastSum = new BigDecimal(0.0);

        //Difference value (totalSum - lastSum)
        BigDecimal delta = new BigDecimal(0.0);
        //Max decimal error allowed between two iterations
        BigDecimal maxError = getMaxError(decimalAccuracy);

        //Counter object that will be used to return the number of iterations
        int counter = 0;
        //This variable holds the factorial value (n!) in the 1/n! computation.
        long nFactorial = 1; //This is initialized at 1 as (0! = 1)

        // Do loop that represents the summation of (1/n!) that is intended to
go
        // from 0 to infinity.
        // This loop will stop when it reaches its intended decimal accuracy,
        // which is int decimalAccuracy.
        do
        {
            // If the loop is just starting (counter = 0), skip the factorial
            // calculation.
            if (counter > 0)
            {
                // Multiply/store current factorial value with updated counter.
                nFactorial = nFactorial * counter;
            }

            //Calculate the current iteration of (1/n!)
            BigDecimal iterationValue = new BigDecimal(1.0 / nFactorial);
            //Add the current iteration value of (1/n!) to our sum (summation
total)
            totalSum = totalSum.add(iterationValue);

            //After the totalSum has been updated, get the difference between
this
            // iteration and the last one.
            delta = totalSum.subtract(lastSum);
            //Update the counter value after a successful iteration.
            counter++;
```

```
            //Store our total summation value in the lastSum variable for
further
            // delta calculations.
            lastSum = totalSum;

            //While delta/difference greater than max allowed error, keep
looping.
        } while (delta.compareTo(maxError) >= 0);
        //After the loop has concluded, print output with calculated E.
        printOutput(lastSum, counter, decimalAccuracy);
    }

    /**
     * This method can be called to print a computed value of e, along with
     * the iterations required to reach such value, and requested decimal
     * precision in the required format.
     *
     * Computed value of e: $calculatedE
     * Expected value of e: 2.7182818284590452 (Depends on decimal accuracy)
     * Required itarations: $counter
     *
     * @param calculatedE Computed value of e
     * @param counter Amount of iterations
     * @param decimalAccuracy Desired decimal precision.
     */
    private static void printOutput(BigDecimal calculatedE, int counter,
                                    int decimalAccuracy)
    {
        // This is the string value of our expected natural number, e, to the
        // desired decimal accuracy for optimal formatted output.
        final String expectedE =
                E_38_DECIMALS.toString().substring(0, decimalAccuracy + 2);

        //Print out with required format.
        System.out.println("Computed value of e: " + calculatedE);
        System.out.println("Expected value of e: " + expectedE);
        System.out.println("Required iterations: " + counter);
    }

    /**
     * This method will return the max error allowed between two iterations
     * of the summation to calculate e to the desired accuracy.
     * @param decimalAccuracy
     * @return returns the max error allowed in BigDecimal form
     */
    private static BigDecimal getMaxError(int decimalAccuracy)
    {
        //This variable holds the max error allowed.
        BigDecimal maxError = new BigDecimal(1.0);

        // This for loop simulates (1)^(-n), where n is decimal accuracy
        // required plus one. The plus one is required as it makes sure that
```

```
      // the decimal of the desired precision doesn't change between
iterations.
      for (int i = 0; i < (decimalAccuracy + 1); i++)
      {
         //Multiply current maxError with 1/10
         maxError = maxError.multiply(BigDecimal.valueOf(1.0 / 10));
      }

      //Return max error allowed between iterations.
      return maxError;
   }
}
```