1.2.15 File input. Develop a possible implementation of the static readInts() method from In (which we use for various test clients, such as binary search on page 47) that is based on the split() method in String.

A computer code with black text

Description automatically generated

Solution: Attached file (FileInput.java + input.txt)

1.2.16 Rational numbers. Implement an immutable data type Rational for rational numbers that supports addition, subtraction, multiplication, and division.

A screenshot of a computer

Description automatically generated

You do not have to worry about testing for overflow (see Exercise 1.2.17), but use as instance variables two long values that represent the numerator and denominator to limit the possibility of overflow. Use Euclid’s algorithm (see page 4) to ensure that the numerator and denominator never have any common factors. Include a test client that exercises all of your methods.

Solution: Attached file (Rational.java)

1.2.17 Robust implementation of rational numbers. Use assertions to develop an implementation of Rational (see Exercise 1.2.16) that is immune to overflow

Solution: Attached file (RobustRational.java)

1.2.18 Variance for accumulator. Validate that the following code, which adds the methods var() and stddev() to Accumulator, computes both the mean and variance of the numbers presented as arguments to addDataValue():A screenshot of a computer code

Description automatically generated

This implementation is less susceptible to roundoff error than the straightforward implementation based on saving the sum of the squares of the numbers.

Solution: Attached file (Accumulator.java + AccumulatorTest.java)

1.2.19 Parsing. Develop the parse constructors for your Date and Transaction implementations of Exercise 1.2.13 that take a single String argument to specify the initialization values, using the formats given in the table below. A computer code with text

Description automatically generated

Solution: Attached file (Parsing.java)