

```

1
2 //Purpose: take input from a file and output the fraction value of the operation
3 //Caspian Peavyhouse
4 //CS101-02
5 //11/23/14
6
7 import java.util.*;
8 import java.io.*;
9
10 public class FractionMath
11 {
12     /*
13         Algorithm for main(args)
14     main(args)
15         File inputFile <-- new File(args[0])
16         Scanner input <-- new Scanner(inputFile)
17
18         File outputFile <-- new File(args[1])
19         FileWriter writerOutput <-- new FileWriter(outputFile)
20         String currentLine <-- new String(input.nextLine())
21
22         writerOutput.write("Fraction Math Version 4")
23         writerOutput.write("Written by Caspian Peavyhouse")
24         writerOutput.write("CS101-02")
25
26         do
27             currentLine <-- currentLine.toLowerCase()
28             if currentLine contains("quit")
29                 break
30             else
31                 readLine(currentLine, writerOutput)
32             currentLine <-- input.nextLine()
33
34         while (input.hasNextLine())
35
36
37     */
38     /*
39
40     Data Table for main
41
42     Variable or Constant      Purpose
43     -----
44     args                      parameter for main
45
46     */
47
48     public static void main(String [] args)throws Exception
49     {
50         File inputFile = new File(args[0]);
51         Scanner input = new Scanner(inputFile);
52
53         File outputFile = new File(args[1]);
54         FileWriter writerOutput = new FileWriter(outputFile);
55         String currentLine = new String(input.nextLine());
56
57         writerOutput.write("FractionMath Version 4\n");
58         writerOutput.write("Written by Caspian Peavyhouse\n");
59         writerOutput.write("CS101-02\n\n");
60
61         do
62         {
63             currentLine = currentLine.toLowerCase();
64             if (currentLine.contains("quit"))
65             {
66                 break;
67             }
68

```

```

69         else
70         {
71             readLine(currentLine, writerOutput);
72         }
73         currentLine = input.nextLine();
74
75     } while (input.hasNextLine());
76
77     writerOutput.close();
78
79 } //main(args)
80
81
82 /*
83 readLine(String currentLine, FileWriter writerOutput)
84
85     writerOutput.write(currentLine + \n)
86
87     currentLine = currentLine.replace('/', ' ')
88     Scanner stringScan <-- new Scanner(currentLine)
89     Scanner numberScan <-- new Scanner(currentLine)
90     if (!(currentLine.charAt(0)>= '0' && currentLine.charAt(0) <= '9')
91         || currentLine.charAt(0) == '-')
92         numberScan.next()
93
94     int firstNum <-- numberScan.nextInt()
95     int secondNum <-- numberScan.nextInt()
96
97     String currentString <-- new String(stringScan.next())
98
99     Fraction currentFraction <-- new Fraction(firstNum, secondNum)
100    Fraction nextFraction <-- new Fraction()
101
102    while (stringScan.hasNext())
103
104        if (currentString.equals("add"))
105            numberScan.next()
106            Fraction nextFraction <-- new Fraction(numberScan.nextInt(),
107                numberScan.nextInt())
108            currentFraction <-- readAdd(writerOutput, currentFraction, nextFraction)
109        else if (currentString.equals("sub"))
110            numberScan.next()
111            Fraction nextFraction <-- new Fraction(numberScan.nextInt(),
112                numberScan.nextInt())
113            currentFraction <-- readSubtract(writerOutput,
114                currentFraction, nextFraction)
115        else if (currentString.equals("mul"))
116            numberScan.next()
117            Fraction nextFraction <-- new Fraction(numberScan.nextInt(),
118                numberScan.nextInt())
119            currentFraction <-- readMultiply(writerOutput, currentFraction,
120                nextFraction)
121        else if (currentString.equals("div"))
122            numberScan.next()
123            Fraction nextFraction <-- new Fraction(numberScan.nextInt(),
124                numberScan.nextInt())
125            currentFraction <-- readDivide(writerOutput, currentFraction, nextFraction)
126        else if (currentString.equals("rec"))
127            writerOutput.write("\tthe reciprocal of " + currentFraction.toString()
128                + " is " + currentFraction.reciprocal() + "\n")
129            currentFraction <-- currentFraction.reciprocal()
130            currentString <-- stringScan.next()
131
132
133 */
134 /*
135     Data Table for readLine
136

```

Variable or Constant	Purpose
args	parameter for readLine
firstNum	storest the value for the firs numerator
secondNum	stores the value for the first denominator

```

137
138
139
140
141
142 */
143     private static void readLine(String currentLine, FileWriter writerOutput)
144         throws Exception
145     {
146         writerOutput.write(currentLine + "\n");
147         currentLine = currentLine.replace('/', ' ');
148         Scanner stringScan = new Scanner(currentLine);
149         Scanner numberScan = new Scanner(currentLine);
150         if (!((currentLine.charAt(0)>= '0' && currentLine.charAt(0) <= '9')
151             || currentLine.charAt(0) == '-'))
152         {
153             numberScan.next();
154         }
155         int firstNum = numberScan.nextInt();
156         int secondNum = numberScan.nextInt();
157
158         String currentString = new String(stringScan.next());
159
160         Fraction currentFraction = new Fraction(firstNum, secondNum);
161
162
163
164
165         while (stringScan.hasNext())
166         {
167
168             if (currentString.equals("add"))
169             {
170                 numberScan.next();
171                 Fraction nextFraction = new Fraction(numberScan.nextInt(), numberScan.nextInt()
172 );
173                 currentFraction = readAdd(writerOutput, currentFraction, nextFraction);
174             }
175             else if (currentString.equals("sub"))
176             {
177                 numberScan.next();
178                 Fraction nextFraction = new Fraction(numberScan.nextInt(), numberScan.nextInt()
179 );
180                 currentFraction = readSubtract(writerOutput, currentFraction, nextFraction);
181             }
182             else if (currentString.equals("mul"))
183             {
184                 numberScan.next();
185                 Fraction nextFraction = new Fraction(numberScan.nextInt(), numberScan.nextInt()
186 );
187                 currentFraction = readMultiply(writerOutput, currentFraction, nextFraction);
188             }
189             else if (currentString.equals("div"))
190             {
191                 numberScan.next();
192                 Fraction nextFraction = new Fraction(numberScan.nextInt(), numberScan.nextInt()
193 );
194                 currentFraction = readDivide(writerOutput, currentFraction, nextFraction);
195             }
196             else if (currentString.equals("rec"))
197             {
198                 writerOutput.write("\tthe reciprocal of " + currentFraction.toString()
199                     + " is " + currentFraction.reciprocal() + "\n");
200                 currentFraction = currentFraction.reciprocal();
201             }
202             currentString = stringScan.next();
203         }
204     }

```

```

201     }//readLine
202
203
204  /*
205  readAdd(FileWriter writerOutput, Fraction currentFraction, Fraction nextFraction)
206      writerOutput.write("\t" + currentFraction.toString() + "\n")
207      writerOutput.write("\tadd " + nextFraction.toString() + " equals "
208          + currentFraction.add(nextFraction) + "\n")
209      currentFraction = currentFraction.add(nextFraction)
210      return currentFraction
211  */
212  /*
213      Data Table for readAdd
214
215  Variable or Constant      Purpose
216  -----
217  args                      parameter for readLine
218
219  */
220  private static Fraction readAdd(FileWriter writerOutput,
221      Fraction currentFraction, Fraction nextFraction) throws Exception
222  {
223      writerOutput.write("\t" + currentFraction.toString() + "\n");
224      writerOutput.write("\tadd " + nextFraction.toString() + " equals "
225          + currentFraction.add(nextFraction) + "\n");
226      currentFraction = currentFraction.add(nextFraction);
227      return currentFraction;
228  }//readAdd
229
230
231  /*
232  readSubtract(FileWriter writerOutput, Fraction currentFraction, Fraction nextFraction)
233      writerOutput.write("\t" + currentFraction.toString() + "\n")
234      writerOutput.write("\tsubtract " + nextFraction.toString() + " equals "
235          + currentFraction.subtract(nextFraction) + "\n")
236      currentFraction = currentFraction.subtract(nextFraction)
237      return currentFraction
238  */
239  /*
240      Data Table for readSubtract
241
242  Variable or Constant      Purpose
243  -----
244  args                      parameter for readLine
245
246  */
247  private static Fraction readSubtract(FileWriter writerOutput,
248      Fraction currentFraction, Fraction nextFraction) throws Exception
249  {
250      writerOutput.write("\t" + currentFraction.toString() + "\n");
251      writerOutput.write("\tsubtract " + nextFraction.toString() + " equals "
252          + currentFraction.subtract(nextFraction) + "\n");
253      currentFraction = currentFraction.subtract(nextFraction);
254      return currentFraction;
255  }//readSubtract
256
257
258  /*
259  readMultiply(FileWriter writerOutput, Fraction currentFraction, Fraction nextFraction)
260      writerOutput.write("\t" + currentFraction.toString() + "\n")
261      writerOutput.write("\tmultiply by " + nextFraction.toString() + " equals "
262          + currentFraction.multiply(nextFraction) + "\n")
263      currentFraction = currentFraction.multiply(nextFraction)
264      return currentFraction
265  */
266  /*
267      Data Table for readMultiply
268

```

```

269 Variable or Constant      Purpose
270 _____
271 args                      parameter for readLine
272
273 */
274 private static Fraction readMultiply(FileWriter writerOutput,
275     Fraction currentFraction, Fraction nextFraction) throws Exception
276 {
277     writerOutput.write("\t" + currentFraction.toString() + "\n");
278     writerOutput.write("\tmultiply by " + nextFraction.toString() + " equals "
279         + currentFraction.multiply(nextFraction) + "\n");
280     currentFraction = currentFraction.multiply(nextFraction);
281     return currentFraction;
282 }//readMultiply
283
284
285 /*
286 readDivide(FileWriter writerOutput, Fraction currentFraction, Fraction nextFraction)
287     writerOutput.write("\t" + currentFraction.toString() + "\n")
288     writerOutput.write("\tdivide by " + nextFraction.toString() + " equals "
289         + currentFraction.divide(nextFraction) + "\n")
290     currentFraction = currentFraction.divide(nextFraction)
291     return currentFraction
292 */
293 /*
294     Data Table for readDivide
295
296 Variable or Constant      Purpose
297 _____
298 args                      parameter for readLine
299
300 */
301 private static Fraction readDivide(FileWriter writerOutput,
302     Fraction currentFraction, Fraction nextFraction) throws Exception
303 {
304     writerOutput.write("\t" + currentFraction.toString() + "\n");
305     writerOutput.write("\tdivide by " + nextFraction.toString() + " equals "
306         + currentFraction.divide(nextFraction) + "\n");
307     currentFraction = currentFraction.divide(nextFraction);
308     return currentFraction;
309 }//readDivide
310
311
312 }//class FractionMath

```