```
 1: /**
 2:  * This class allows the user to process a datafile to fix incorrectly inputted
 3:  * names. It assumes a file format where headers (which remain untouched) are
 4:  * marked with pound signs at the beginning of the line. Blank lines are
 5:  * maintained. The only lines which are changed are those containing names in
 6:  * the format 'firstname,lastname' which are then corrected to the format
 7:  * 'lastname, firstname'.
 8:  *
 9:  * The user can call the program WITHOUT command-line arguments which will
10:  * default to the input filename 'datafile' and output filename 'datafile.fix'.
11:  *
12:  * The user can also call the program WITH command-line arguments where the
13:  * first is the name of the file that is to be processed and the second is that
14:  * of the file that is to be outputted to.
15:  *
16:  * NOTE: This program is not destructive and will die if the output file
17:  * already exists and will not overwrite files. It leaves the input file intact.
18:  *
19:  * The program can be modified in the future to get rid of hard-coded filenames
20:  * and instead to ask for the filenames interactively. It can also be modified
21:  * to ask the user for permission to overwrite an existing file.
22:  *
23:  * @name    NameSwapper
24:  * @author  Ravi S. Ramphal
25:  * @class   CCSF CS111B
26:  * @date    2017.07.06
27:  * @version 1.0
28:  */
29:
30: import java.io.*;
31: import java.util.*;
32:
33: public final class NameSwapper
34: {
35:     /**
36:      * This is a die method that prints a message to the stderr stream and
37:      * exits the program with an error code.
38:      *
39:      * @param errorMessage The error message to be displayed to users
40:      */
41:     private static void die(String errorMessage)
42:     {
43:         System.err.println(errorMessage);
44:         System.exit(1);
45:     }
46:
47:     /**
48:      * This method validates the input file for the program to work with and
49:      * returns its 'File' object. It will cause the program to die if:
50:      *     - the input name is not provided
51:      *     - the file does not exist exists
52:      *     - the file is not a file
53:      *     - the file is a directory
54:      *     - the file does exist and is a file, but cannot be read
55:      *
56:      * @param  filename  A string containing the name of the input file
57:      * @return File      The File object of the file with the name provided
58:      */
59:     private static File validateInput(String filename)
60:     {
61:         if (filename.length() < 1) die("Input filename not provided!");
62:
63:         File file = new File(filename);
64:
65:         if (!file.exists())      die(filename + " does not exist!");
66:         if (!file.isFile())      die(filename + " is not a file!");
67:         if ( file.isDirectory()) die(filename + " is a directory!");
68:         if (!file.canRead())     die(filename + " cannot be read!");
```

```
69:
70:         return file;
71:     }
72:
73:     /**
74:      * This method creates the output file for the program to work with and
75:      * returns its 'File' object. It will cause the program to die if:
76:      *     - the output name is not provided
77:      *     - the file already exists
78:      *     - the file could not be created
79:      *     - the file could be created, but not written to
80:      *
81:      * @param  filename  A string containing the name of the desired output file
82:      * @return File      The File object of the file with the name provided
83:      */
84:     private static File createOutput(String filename)
85:     {
86:         if (filename.length() < 1) die("Output filename not provided!");
87:
88:         File file = new File(filename);
89:
90:         if (file.exists()) die(filename + " already exists!");
91:
92:         try
93:         {
94:             file.createNewFile();
95:
96:             if (!file.canWrite()) die(filename + " cannot be written to!");
97:         }
98:         catch (IOException e)
99:         {
100:            die(filename + " could not be created!");
101:        }
102:
103:        return file;
104:    }
105:
106:    /**
107:     * The method checks whether or not the line is a header.
108:     * It defaults to 'false' and will return 'false' on an empty string.
109:     * Otherwise, if the line begins with a pound sign, it returns 'true'.
110:     *
111:     * @param  line    The string containing the line to be processed
112:     * @return boolean A boolean value as to whether the line is a header or not
113:     */
114:    private static boolean isHeader(String line)
115:    {
116:        if (line.length() == 0) return false;
117:        if (line.charAt(0) == '#') return true;
118:        return false;
119:    }
120:
121:    /**
122:     * This method is responsible for processing each line.
123:     *
124:     * The line is returned unmodified if:
125:     *     - it is a header
126:     *     - there are no commas
127:     *     - there is more than one comma
128:     *
129:     * Otherwise, the line is returned swapped with a comma and a space between.
130:     *
131:     * @param  line   The string containing the line to be processed
132:     * @return String The processed string
133:     */
134:    private static String processLine(String line)
135:    {
136:        if (isHeader(line)) return line;          // do not change header
```

```
137:         if (line.indexOf(",") == -1) return line; // only change comma lines
138:
139:         String[] parts = line.split(",");
140:
141:         if (parts.length > 2) return line;        // too many parts to process
142:
143:         return parts[1] + ", " + parts[0];
144:     }
145:
146:     /**
147:      * This method takes in the two files (already validated). Technically,
148:      * error handling is not necessary in this method since the input and
149:      * output files would have already been validated in the main method.
150:      *
151:      * A 'Scanner' object is created to iterate over each line of the input file
152:      * and each line is processed and then written to the output file.
153:      *
154:      * @param inputFile  The File that is to be processed
155:      * @param outputFile The File that is to be outputted to
156:      */
157:     private static void processFile(File inputFile, File outputFile)
158:     {
159:         try
160:         {
161:             Scanner     scanner     = new Scanner(inputFile);
162:             FileWriter  fileWriter  = new FileWriter(outputFile, true);
163:             PrintWriter printWriter = new PrintWriter(fileWriter);
164:
165:             String line;
166:
167:             while(scanner.hasNext())
168:             {
169:                 line = scanner.nextLine();
170:                 printWriter.println(processLine(line));
171:             }
172:
173:             scanner.close();
174:             printWriter.close();
175:
176:             System.out.println("File processed.");
177:         }
178:         catch (FileNotFoundException e)
179:         {
180:             die("Input file not found!");
181:         }
182:         catch (IOException e)
183:         {
184:             die("Output file not found!");
185:         }
186:     }
187:
188:     /**
189:      * This is the 'main' method of this class. It sets up default filenames
190:      * but also allows for names to be passed in through command-line arguments.
191:      * File validation is done here and then the file is processed.
192:      *
193:      * @param array An array of arguments provided to the program
194:      */
195:     public static void main (String ... args)
196:     {
197:         String input  = "datafile";
198:         String output = "datafile.fix";
199:
200:         if (args.length == 2)
201:         {
202:             input  = args[0];
203:             output = args[1];
204:         }
```

```
205:
206:         File inputFile  = validateInput(input);
207:         File outputFile = createOutput(output);
208:
209:         processFile(inputFile, outputFile);
210:     }
211: }
```