Module Nine - Assignment Submission

The Assignment

> Supplied is a data file from the US Census, https://www.census.gov/, which contains data from US school districts and reports statistics related to child poverty. It is desired to have a summary report which calculate basic statistics at the state level.

> Desired Implementation: Java implementation to read the supplied text data and produce a report similar to the below:

> There should be two separate “programs” (main()), one to read the text data file and write a reformatted file to be read by the second program which will create the report to standard out. The format of the reformatted file can be any form such as text, binary, csv, Serialized, JSON, html, or other form of your choosing. Note before the report is displayed, a single line with “File: “ then the path of the input file for the report is displayed. The first program will have 3 run-time parameters passed into the program via the command line, the data source file path, the destination file path, and the number of records in the data file, SmallAreaIncomePovertyEstData. The second program will have 2 run-time parameters, the input file path and the number of records.

> The programs should use standard Java (SE) code and compile without errors or warnings. It should also run without errors or warnings when given valid input. The programs should provide reasonable parameter validation (correct number of parameters, reasonable values, etc.). The file produced by the first program should not be deleted after running the report program. The program's code should be reasonable formatted and commented as demonstrated so far in the course

Design

Coming from the last set of requirements, it sees that we'll be needing to main functions:

1. An I/O which parses the text files and shunts it into a more easily parsable format.
2. The actual print and display function

To do the reading, since we're working with analysis on a line by line basis, we'll probably want to use a buffered reader/writer in order to handle the filei/o

To do this, we can use the same general file i/o, with the first program having the only instance of O from the I/O

We'll then do reformatting in the first file, which is just string manipulation to parse the data, and then write out a buffer.

To store the data during parsing, I'm intending to store the relevant output data in a struct, which we index by the state code since that's the compression we want for the data.

The second file will just coherently print the data out.

It appears, from the screenshots, that there is a spacing requirement which has some notion of rightward alignment of the data.

Eyeballing it, we'll have the following:

field	num chars
state	5
population	10
Child Population	17
Child Poverty Population	24
% Child Poverty	14

I think a CSV is the most akin to a table for our purposes, so I'll use a csv to write out to.

Implementation

```
package module10;

import java.io.*;

// Class which parses text data from a census file
// Program takes in file path for input, file path for output,
// and the number of records in the file
public class ParseCensus {
    // Program Entry Point
    static class CensusData {
        int population;
        int childPopulation;
        int childPovertyPopulation;

        CensusData() {
            population = 0;
            childPopulation = 0;
            childPovertyPopulation = 0;
        }
    }
    public static void main(String[] args) {
        // Check if the correct number of arguments is provided
        if (args.length != 3) {
            System.out.println("Usage: java ParseCensus <input_file>
<output_file> <num_records>");
            return;
        }

        String inputFilePath = args[0];
        String outputFilePath = args[1];
        int numRecords = Integer.parseInt(args[2]);

        // Array of CensusData objects to store the census data
        CensusData[] censusDataArray = new CensusData[56];
        for (int i = 0; i < censusDataArray.length; i++) {
            censusDataArray[i] = new CensusData();
        }

        // Start the loop to process the census data

        try (BufferedReader reader = new BufferedReader(new FileReader(inputFilePath));) {
            // Read each line from the input file
            String line;
            int recordCount = 0;
            while ((line = reader.readLine()) != null && recordCount < numRecords) {
                // process line and update data structure
                parseCensusData(line, censusDataArray);
                recordCount++;
            }
        } catch (IOException e) {
            System.err.println("Error reading or writing files: " + e.getMessage());
        }

        // Write the census data to json
        writeToFile(outputFilePath, censusDataArray);
    }

    // Method to parse a line of census data and update the censusDataArray
    static void parseCensusData(String line, CensusData[] censusDataArray) {
        String[] data = line.split("\\s+"); // Assuming space-separated values


        // Convert first entry to int
        int state = Integer.parseInt(data[0]) - 1;
        // Get the population, child population, and child poverty population
        // from the data array using "negative indexing"
        int populationPointer = (data.length - 5);

        int population = Integer.parseInt(data[populationPointer]);
        int childPopulation = Integer.parseInt(data[populationPointer + 1]);
        int childPovertyPopulation = Integer.parseInt(data[populationPointer + 2]);

        censusDataArray[state].population += population;
        censusDataArray[state].childPopulation += childPopulation;
        censusDataArray[state].childPovertyPopulation += childPovertyPopulation;
    }

    // Method to write the census data a csv file
    static void writeToFile(String outputFilePath, CensusData[]
censusDataArray) {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(outputFilePath))) {
            // Write the header

writer.write("State,Population,ChildPopulation,ChildPovertyPopulation");
            writer.newLine();

            // Write the census data
            for (int i = 0; i < censusDataArray.length; i++) {
                CensusData data = censusDataArray[i];
                writer.write((i + 1) + "," + data.population + "," +
data.childPopulation + "," + data.childPovertyPopulation);
                writer.newLine();
            }
        } catch (IOException e) {
            System.err.println("Error writing to file: " + e.getMessage());
        }
    }
}

package module10;

import java.io.*;
import java.nio.file.*;
import java.util.List;
import java.text.NumberFormat;
import java.util.Locale;

// Class which takes in csv representation of census data and prints it
// to the console
public class PrintData {
    // Program Entry Point
    public static void main(String[] args) {
        // Check if the correct number of arguments is provided
        if (args.length != 2) {
            System.out.println("Usage: java PrintData <input_file>
<num_records>");
            return;
        }

        // Extract Args
        String inputFilePath = args[0];

        // Print Header
        printHeader();

        // Read CSV file and print data
               try {
            List<String> lines =
Files.readAllLines(Paths.get(inputFilePath));
            processCsv(lines);
        } catch (IOException e) {
            System.err.println("Error reading CSV file: " +
e.getMessage());
        }
    }

    // Method to process CSV data
    public static void processCsv(List<String> lines) {
        // Process each line in the CSV, except the header
        NumberFormat numberFormat = NumberFormat.getInstance(Locale.US);
        if (lines.isEmpty()) return; // No data to process
        lines.remove(0); // Remove header line
        for (String line : lines) {
            String[] columns = line.split(",");
            if (columns.length != 4) continue; // Skip invalid rows

            try {
                int state = Integer.parseInt(columns[0].trim());
                int population = Integer.parseInt(columns[1].trim());
                int childPopulation = Integer.parseInt(columns[2].trim());
                int childPovertyPopulation =
Integer.parseInt(columns[3].trim());

                double povertyPercentage = ((double) childPovertyPopulation
/ childPopulation) * 100;

                // Print data rows with right justification
                System.out.printf("%5s    %10s %17s %24s %16.2f%n",
                    String.format("%02d", state),
                    numberFormat.format(population),
                    numberFormat.format(childPopulation),
                    numberFormat.format(childPovertyPopulation),
                    povertyPercentage);
            } catch (NumberFormatException e) {
                System.err.println("Error parsing numbers in line: " +
line);
            }
        }
    }

    // Method to print header
    static void printHeader() {
        // Print header
        System.out.print("State");
        System.out.print("    Population");
        System.out.print("  Child Population");
        System.out.print(" Child Poverty Population");
        System.out.print("  % Child Poverty");
        System.out.println();
        // Print Table Ticks
        System.out.print("-----");
        System.out.print("    ----------");
        System.out.print("  ----------------");
        System.out.print(" ------------------------");
        System.out.print("  ---------------");
        System.out.println();
    }
}
```

Output:

```
State    Population  Child Population Child Poverty Population  % Child
Poverty
-----    ----------  ---------------- ------------------------  -----------
----
   01     4,833,722           814,377                  205,023
25.18
   02       735,132           132,740                   16,118
12.14
   03             0                 0                        0
NaN
   04     8,688,149         1,182,931                  288,777
24.41
   05     2,959,373           516,950                  132,920
25.71
   06    48,909,205         6,667,268                1,468,715
22.03
   07             0                 0                        0
NaN
   08     5,268,367           902,796                  139,381
15.44
   09     3,747,676           593,629                   77,895
13.12
   10       925,749           147,239                   25,169
17.09
   11       646,449            70,507                   20,544
29.14
   12    19,552,860         2,948,361                  678,022
23.00
   13    10,010,465         1,821,201                  445,608
24.47
   14             0                 0                        0
NaN
   15     1,404,054           216,496                   29,375
13.57
   16     1,612,136           314,294                   56,633
18.02
   17    17,704,060         2,224,288                  427,235
19.21
   18     6,570,099         1,165,146                  226,599
19.45
   19     3,090,416           529,306                   77,634
14.67
   20     2,893,957           523,686                   84,325
16.10
   21     4,405,846           739,127                  171,418
```

```
23.19
   22     4,625,470           804,740                  212,904
26.46
   23     1,358,717           196,262                   31,174
15.88
   24     5,928,814           977,312                  120,049
12.28
   25     7,107,877         1,028,400                  153,286
14.91
   26     9,895,622         1,672,433                  351,702
21.03
   27     5,420,550           931,542                  119,437
12.82
   28     2,991,207           539,006                  170,629
31.66
   29     6,044,171         1,020,848                  203,216
19.91
   30     1,913,094           162,709                   30,655
18.84
   31     1,868,516           334,188                   49,030
14.67
   32     2,790,136           483,411                   99,599
20.60
   33     1,472,055           205,461                   19,714
9.60
   34    10,552,547         1,488,882                  222,992
14.98
   35     2,085,287           368,816                  103,790
28.14
   36    19,901,043         3,066,336                  666,553
21.74
   37     9,848,060         1,673,310                  386,419
23.09
   38       723,393           113,921                   12,685
11.13
   39    11,570,743         1,958,998                  398,688
20.35
   40     3,851,487           682,548                  144,867
21.22
   41     3,931,430           627,584                  118,023
18.81
   42    12,773,801         1,999,741                  342,181
17.11
   43             0                 0                        0
NaN
   44     1,065,907           159,355                   31,368
19.68
   45     4,790,785           787,482                  194,639
24.72
   46       844,877           148,002                   24,675
16.67
   47     6,778,703         1,091,900                  260,103
23.82
   48    26,452,422         5,101,161                1,198,322
```

```
23.49
   49     2,900,872           642,722                   85,745
13.34
   50       940,840            92,223                   11,990
13.00
   51     8,260,405         1,352,420                  190,734
14.10
   52             0                 0                        0
NaN
   53     6,971,406         1,151,175                  197,126
17.12
   54     1,854,304           279,484                   64,539
23.09
   55     5,956,920           963,445                  157,356
16.33
   56       580,850            99,034                   11,684
11.80
```