Technical Assessment for AI intern role – Smart Data Solutions – JAVA PROGRAMMING EXERCISE

SUBMITTED BY – NANDAKISHORE B

For this task, the goal is to combine two CSV files into one output file, but only for the rows where the first column (ID) matches in both files. If a match is found, we keep the ID from CSV1, then the rest of CSV1's columns, and then all of CSV2's columns except its first column. Rows without a matching ID in the other file are ignored. Although this sounds like a simple "match and join" job, the way we search for matches can hugely affect performance when the files are large.

I explored three different approaches and found that the HashMap method is the most efficient. In the HashMap approach, we first read CSV2 into a map where the key is the ID (first column) and the value is the rest of the row. Then we go through CSV1 line by line, check if the ID exists in the map, and if so, merge the data and write it to the output. This approach has O(n + m) time complexity because each file is read once, and lookups are O(1) on average. Space complexity is O(m) since we only store CSV2 in memory. This is the approach I decided to use because it's both clean and fast for large datasets.

The second approach I considered was sorting both CSV files by the ID column and then merging them using a two-pointer technique, similar to merge sort. Once both files are sorted, matching becomes easy as you move through them in sync. This runs in O(n log n + m log m) due to sorting, and requires storing both files in memory. While this is better than brute force for bigger files, it's still slower than the HashMap method because of the sorting step.

The third and least efficient approach was brute force matching. Here, every row from CSV1 is compared with every row from CSV2 to check for a match. This has a time complexity of O(n × m), making it impractical for larger files. The only advantage is that it's easy to understand and implement, but the speed penalty is huge when datasets grow.

In short, I chose the HashMap method because it's the fastest and scales really well. The logic is straightforward: load CSV2 into a map, process CSV1 while checking for matches, and write the merged results to the output. BufferedReader and BufferedWriter are used for efficient file operations, try-with-resources ensures files close automatically, and exceptions are caught to prevent crashes if there are file issues. This keeps the program efficient, safe, and reliable.

```java
//Approach 1 - HashMap Lookup
/*
This program basically reads two csv files and checks if the first column (ID) matches in both
files.
If it finds a match, it will join that line from csv1 with the rest of the columns from csv2
(excluding the id again).
I used a HashMap to store csv2 data so that searching for matches is super quick.
Then I just loop through csv1 and whenever there's a match, I write the combined line into a
new output file.
The output only contains rows that had matches in both files, so no unmatched junk is written.
Pretty straightforward but does the job neatly.
*/

//Time Complexity   - O(n + m)
// Space Complexity-  O(m)

import java.io.*;
import java.util.*;

public class CSVCombiner {

    // This method will "stick" matching lines from csv1 and csv2 together
    static void combineFiles(String file1, String file2, String outFile) {
        // map to store data from csv2 (excluding the id column)
        HashMap<String, String> map2 = new HashMap<>();

        // 1. Read csv2 and save rows in the map
        try (BufferedReader br = new BufferedReader(new FileReader(file2))) {
            String lineFromCsv2;
            while ((lineFromCsv2 = br.readLine()) != null) {
                String[] bits = lineFromCsv2.split(",");
                if (bits.length > 1) {
                    // join the rest of the columns back into a string
                    String restOfLine = String.join(",", Arrays.copyOfRange(bits, 1,
bits.length));
                    map2.put(bits[0], restOfLine);
                }
            }
        } catch (IOException e) {
            System.out.println("Error reading file 2: " + e.getMessage());
            return;
        }

        // 2. Read csv1 and if there's a match in csv2, write combined row to output
        try (
                BufferedReader br = new BufferedReader(new FileReader(file1));
                BufferedWriter bw = new BufferedWriter(new FileWriter(outFile))
        ) {
            String lineFromCsv1;
            while ((lineFromCsv1 = br.readLine()) != null) {
                String[] bits = lineFromCsv1.split(",");
                String id = bits[0];

                if (map2.containsKey(id)) {
                    // matched → merge csv1 line + csv2 line (minus id)
                    String merged = lineFromCsv1 + "," + map2.get(id);
                    bw.write(merged);
                    bw.newLine();
                }
            }
        } catch (IOException e) {
            System.out.println("Error reading file 1 or writing output: " + e.getMessage());
        }
    }

    public static void main(String[] args) {
        // file names (make sure they're in the project root)
        String csv1 = "csv1.csv";
        String csv2 = "csv2.csv";
        String output = "output.csv";

        // call our cool method
        combineFiles(csv1, csv2, output);

        System.out.println("Done! Check the file: " + output);
    }
}
```

```java
//Approach 2 - Sort + Merge (Two Pointer Method)
/*
Here I decided to sort both CSV files by their ID column first and then walk through them
together using two pointers. This way, I can match rows without checking every single
pair like in the brute force method. Once the IDs match, I join the lines and save them
to the output. It's a nice middle ground between simple and efficient, but it does
require loading and sorting the whole files before matching.
*/
//Time Complexity  -    O(n log n + m log m)
// Space Complexity -   O(n + m)

import java.io.*;
import java.util.*;

public class CSVSortMerge {

    public static void combineFiles(String csv1, String csv2, String output) {
        List<String[]> list1 = new ArrayList<>();
        List<String[]> list2 = new ArrayList<>();

        // Load CSV1
        try (BufferedReader br = new BufferedReader(new FileReader(csv1))) {
            String line;
            while ((line = br.readLine()) != null) {
                list1.add(line.split(","));
            }
        } catch (IOException e) {
            System.out.println("Error reading " + csv1);
        }

        // Load CSV2
        try (BufferedReader br = new BufferedReader(new FileReader(csv2))) {
            String line;
            while ((line = br.readLine()) != null) {
                list2.add(line.split(","));
            }
        } catch (IOException e) {
            System.out.println("Error reading " + csv2);
        }

        // Sort by ID (first column)
        Comparator<String[]> byId = Comparator.comparing(a -> a[0]);
        list1.sort(byId);
        list2.sort(byId);

        // Merge using two pointers
        int i = 0, j = 0;
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(output))) {
            while (i < list1.size() && j < list2.size()) {
                int cmp = list1.get(i)[0].compareTo(list2.get(j)[0]);
                if (cmp == 0) {
                    String merged = String.join(",", list1.get(i)) + "," +
                            String.join(",", Arrays.copyOfRange(list2.get(j), 1,
list2.get(j).length));
                    bw.write(merged);
                    bw.newLine();
                    i++;
                    j++;
                } else if (cmp < 0) {
                    i++;
                } else {
                    j++;
                }
            }
        } catch (IOException e) {
            System.out.println("Error writing output");
        }
    }

    public static void main(String[] args) {
        combineFiles("csv1.csv", "csv2.csv", "output sortmerge.csv");
        System.out.println("Sort + Merge method done.");
    }
}
```

```java
//Approach 3 - Nested Loops
/*
In this version I didn't use any fancy data structures, just pure looping.
First I read both CSV files into two lists and then compared every row from the first file
with every row from the second file to see if their IDs match. When a match is found,
I merge the data and write it to the output. It's pretty straightforward and easy to
understand, but not the fastest thing in the world since it checks every possible pair.
Still works fine for small files.
*/

//Time Complexity   - O(n × m)
// Space Complexity - O(n + m)


import java.io.*;
import java.util.*;

public class CSVNestedLoop {

    public static void combineFiles(String csv1, String csv2, String output) {
        List<String[]> list1 = new ArrayList<>();
        List<String[]> list2 = new ArrayList<>();

        // Load CSV1 into list1
        try (BufferedReader br = new BufferedReader(new FileReader(csv1))) {
            String line;
            while ((line = br.readLine()) != null) {
                list1.add(line.split(","));
            }
        } catch (IOException e) {
            System.out.println("Error reading " + csv1);
        }

        // Load CSV2 into list2
        try (BufferedReader br = new BufferedReader(new FileReader(csv2))) {
            String line;
            while ((line = br.readLine()) != null) {
                list2.add(line.split(","));
            }
        } catch (IOException e) {
            System.out.println("Error reading " + csv2);
        }

        // Compare every row from csv1 with every row from csv2
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(output))) {
            for (String[] row1 : list1) {
                for (String[] row2 : list2) {
                    if (row1[0].equals(row2[0])) {
                        String merged = String.join(",", row1) + "," +
                                String.join(",", Arrays.copyOfRange(row2, 1, row2.length));
                        bw.write(merged);
                        bw.newLine();
                        break; // stop after finding first match
                    }
                }
            }
        } catch (IOException e) {
            System.out.println("Error writing output");
        }
    }

    public static void main(String[] args) {
        combineFiles("csv1.csv", "csv2.csv", "output_nested.csv");
        System.out.println("Nested loop method done.");
    }
}
```

CODE WORKING – program is designed to merge two CSV files, csv1.csv and csv2.csv, based on a common identifier in the first column of each file. It creates a new file, output.csv, containing the merged data.

The program's main logic is contained within the combineFiles method, which follows these steps:

1. **Read csv2.csv and create a map.** The program reads csv2.csv line by line. For each line, it splits the data by the comma delimiter. The first value (the ID) is used as a key, and the rest of the line is stored as a value in a HashMap. This map, named map2, essentially acts as a lookup table, allowing for fast retrieval of data from csv2 using the ID.
2. **Read csv1.csv, find matches, and write to output.** The program then reads csv1.csv line by line. For each line, it extracts the ID from the first column. It checks if this ID exists as a key in the map2 from the previous step.
    - If a match is found, it combines the current line from csv1 with the corresponding value from map2. This combined string is then written as a new line to the output.csv file.
    - If no match is found, the line from csv1 is ignored.

The main method simply defines the file names and calls the combineFiles method to perform the merging operation. The program uses try-with-resources blocks to ensure that the file readers and writers are automatically closed, even if an error occurs.