Command:

```
java -classpath .\Mini-Campaign\target\classes\ Main <path-of-CSV-File-1> <path-of-CSV-File-2> [flags]

flags:

-d Delimiters used in Both CSV Files (Default: “,”)

-p Columns Present in Both CSV Files (Default: “true”)

-c Unique Column Combination (Default: All Columns across both CSV Files are compared)
```

Equivalence Class Partitioning with Boundary Value Analysis:

1. A possible class of inputs to evaluate is comparing 2 CSV files with the same content values. This could be done by providing the same file path twice in the command line. The expected output would be that there would be no mismatches. The rationale is to check whether the program generates an output indicating no mismatches.
    a. Middle Value: “file1.csv” “file1.csv”
    b. Boundary Value: “file1.csv” “file2.csv” where all row entry and column values in file1.csv and file2.csv are exactly the same but the columns are ordered differently (e.g. “ID”,“Currency”,“Type” in file1.csv but “ID”,“Type”,“Currency”, in file2.csv).

2. A possible class of inputs to evaluate is providing invalid file path for either one of the 2 files (File should not exist). The expected output would be that an exception is generated – FileNotFoundException. The rationale is to check whether the program can catch the exception and safely terminate the program.
    a. Middle Value: “file1.csv” “fileR.csv” where fileR.csv does not exist
    b. Boundary Value: “file1.csv” “file.csv” where file.csv does not exist

3. A possible class of inputs to evaluate is providing a file format that is not CSV for either one of the 2 files (File should not exist). The expected output would be that an exception is generated – IllegalArgumentException. The rationale is to check whether the program parses only CSV Files for comparison.
    a. Middle Value: “file1.txt” “file2.xlsx”
    b. Boundary Value: “file1.csv” “file2.tsv”

4. A possible class of inputs to evaluate is providing an invalid set of flags (Possible examples: -p -p, -p null, -d “comma”, etc.). The expected output would be that an exception is generated – IllegalArgumentException. The rationale is to check whether the program can capture invalid arguments in the program.
    a. Middle Value: “file1.csv” “file2.csv” -d “comma” -p “null”
    b. Boundary Value: “file1.csv” “file2.csv” -d -p “true”

5. A possible class of inputs to evaluate is providing a CSV File with Invalid Columns (e.g. empty column name – “Currency”, “”, “Type” or duplicate column name – “Currency”, “Currency”, “Type”). The expected output would be that a custom exception is generated – CSVException. The rationale is to check whether the program can catch these invalid column names before performing the comparison to generate mismatches.
    a. Middle Value: “file1.csv” “file2.csv” -c “Currency”, “Type” where columns “Currency” and “Type” exists in both CSV Files but one of the files have 2 columns named “Currency”
    b. Boundary Value: “file1.csv” “file2.csv” where an empty column name exists in both CSV Files

6. A possible class of inputs to evaluate is providing a CSV File where there exists a row whose entries count do not match the total number of columns present. The expected output would be that a custom exception is generated – CSVException. The rationale is to check whether the program can check whether there is extraneous row entries in the CSV File that may hinder the comparison.
    a. Middle Value: “file1.csv” “file2.csv” where there is an additional entry in the row of values (excluding the column name rows) in both CSV Files
    b. Boundary Value: “file1.csv” “file2.csv” where there is one less entry in all the rows of values (excluding the column name rows) in both CSV Files. (This is a boundary because by adding a comma to indicate null entries in all rows of values. The program should execute successfully.)

7. A possible class of inputs to evaluate is providing a Unique Combination (Columns Selected for Comparison) where the provided column name is not present in both

CSV Files. The expected output would be that a custom exception is generated – CSVException. The rationale is to check whether the program can detect whether the unique combination provide a common basis of comparison between the 2 CSV Files.

   a. Middle Value: “file1.csv” (with column names “Currency”, “Type”) “file2.csv” (with column names “Account ID, Balance”) -c “Balance”,“Type”
   b. Boundary Value: “file1.csv” (with column names “Currency”, “Account ID”,“Type”) “file2.csv” (with column names “Account ID”, “Balance”) -c “Balance”,“Account ID” (This is considered a Boundary Value because the program should execute successfully if “Balance” is removed from the Unique Combination.)

8. A possible class of inputs to evaluate is providing 2 CSV Files with at least one different column name present and performing a full comparison. The expected output would be that a custom exception is generated – CSVException. The rationale is to check whether the program can detect whether the CSV Files can be compared row-wise as a whole.
   a. Middle Value: “file1.csv” (with columns “Account No.”,“ID”) “file2.csv” (with columns “Balance”,“Type”,“Currency”)
   b. Boundary Value: “file1.csv” (with columns “Balance”,“Type”,“ID”) “file2.csv” (with columns “Balance”,“Type”,“Currency”) (This is considered a Boundary Value because the columns ID and Currency are not present in both CSV Files. Changing the column name “Currency” to “ID” would result in the successful execution of the program.)

9. A possible class of inputs to evaluate is providing a CSV File with duplicate rows present but the row does not exist in the other CSV File and performing a full comparison. The expected output would be that a warning is generated but only 1 unique mismatch is generated. The rationale is to check whether the program is able to detect duplicates in the CSV Files and treat them as 1 unique set when comparing with the other file.
   a. Middle Value: “file1.csv” “file2.csv” where “file1.csv” contain 1 unique set of N duplicate rows that is not present in “file2.csv”
   b. Boundary Value: “file1.csv” “file2.csv” where each CSV file contain N unique set of duplicate rows that are not present in each other. (This is

considered a Boundary Value as if overlapping duplicate rows across both CSV Files can result in less than N unique mismatches.)

10. A possible class of inputs to evaluate is providing a CSV File with duplicate rows present and the row exists in the other CSV File and performing a full comparison. The expected output would be that a warning is generated but no mismatch is generated. The rationale is to check whether the program is able to detect duplicates in the CSV Files and treat them as 1 unique set when comparing with the other file.
    a. Middle Value: “file1.csv” “file2.csv” where “file1.csv” contain 1 unique set of N duplicate rows but “file2.csv” contains more than 1 such rows.
    b. Boundary Value: “file1.csv” “file2.csv” where “file1.csv” contain 1 unique set of N duplicate rows but “file2.csv” contains only 1 such row. (This is considered a Boundary Value as the removal of the row in “file2.csv” can result in a different outcome.)

11. A possible class of inputs to evaluate is providing a CSV File with the same column names and unique rows across both files. The expected output would be that all rows would be generated as a mismatch. The rationale is to check whether the program is able to perform a successful comparison between 2 CSV files.
    a. Middle Value: “file1.csv” “file2.csv” where all entries within the rows (excluding column name row) are unique and are not present across the 2 CSV files.
    b. Boundary Value: “file1.csv” “file2.csv” where only 1 entry within each row (excluding column name row) is modified to be unique and are not present across the 2 CSV files.