ESC Week 9
Name: Parwani Jayati
Student ID: 1005622

Use Case 1- Receive CSV 1
Input space- file upload

Rationale:
It is important to test out which files format are uploaded by the user.
Invalid formats or empty or no files upload should fail the test case, while only a valid csv should pass the test case.

Equivalence Class – Partition	Input Value-Type	Expected Outcome
No file uploaded	Boundary Value	Invalid
Wrong format file uploaded (pdf, png etc)	Middle Value	Invalid
Empty csv file uploaded	Boundary Value	Invalid
Csv file with only one row- headers	Boundary Value	Valid
Csv file within max row (java) limit	Middle Value	Valid
Csv file that is 1 less than max row (java) limit	Boundary Value	Valid

Use Case 2- Receive CSV 2
Input space- file upload

Rationale:
It is important to test out the second file- which contains the unique combination to be compared against.
Invalid formats or empty or no files upload should fail the test case, while only a valid csv should pass the test case.

Equivalence Class – Partition	Input Value-Type	Expected Outcome
No file uploaded	Boundary Value	Invalid
Wrong format file uploaded (pdf, png etc)	Middle Value	Invalid
Empty csv file uploaded	Boundary Value	Invalid
Csv file with only one row – headers	Boundary Value	Valid
Csv file within max row (java) limit	Middle Value	Valid
Csv file that is 1 less than max row (java) limit	Boundary Value	Valid

Use case 3- Validate both csv files
Input Space: 5 column headers in a csv: [Customer ID#, Account No., Currency, Type, Balance, Transaction Date]
Rationale:
Content of the Csv files must be validated before csv comparison, so equivalence class input space has to be the valid header of the csv files uploaded as given above in blue.

Equivalence Class – Partition	Input Value-Type	Expected Outcome
Csv file with missing column headers	Boundary Value [Customer ID#, Account No]	Invalid
Csv file with 5 columns but wrong column headers	Middle Value [Customer ID#, Account No., Currency, Type, Date]	Invalid
Csv file with all 5 expected column headers but with an extra new column header	Boundary Value [Customer ID#, Account No., Currency, Type, Balance, Transaction Date]	Invalid
Csv file 5 expected column headers but with an extra duplicate column header	Boundary Value [Customer ID#, Account No., Currency, Type, Balance, Type]	Valid
Csv file with 5 column headers as expected: Customer ID#, Account No., Currency, Type, Balance	Middle Value [Customer ID#, Account No., Currency, Type, Balance]	Valid
Csv file with 5 correct column headers in a different order	Boundary Value [Customer ID#, Account No., Currency, Balance, Type]	Valid

Use Case 4- Parse csv row by row
Input Space: [**ID1**, BOS963211, USD, SAVINGS, 962510]

Rationale:
Content of the Csv files must be validated as each row is parsed and invalid rows to be removed. Thus, the partitioning includes content of each row as we parse in an array list. Different cases have been simulated.

Equivalence Class – Partition	Input Value-Type	Expected Outcome
Csv file with missing details in a row	Boundary Value [**ID1**, BOS963211, USD, 962510] (e.g. Missing account type)	Invalid
Csv file with duplicate account number or customer id in a row	Middle Value [**ID1**, BOS963211, USD, 962510]	Invalid

	[**ID1**, BOS963211, USD, 962510] (e.g. id 1 repeated)	
Csv file with an extra detail in a row	Boundary Value [**ID1**, BOS963211, USD, 962510, 27/06]	Invalid
Csv file with a repeated detail in a row	Boundary Value [**ID1**, BOS963211, USD, 962510, USD]	Valid
Csv file with all details in every row	Middle Value [**ID1**, BOS963211, USD, SAVINGS, 962510]	Valid
Csv file with all details in a row but row order is different	Boundary Value [**ID1**, USD,BOS963211, SAVINGS, 962510]	Valid

Use Case 5- Compare csv files row by row
Input Space: **2 corresponding array lists , one from each csv**

Rationale:
Row by row comparison is the main aim of the code and hence, its partitioning involves various cases which can be simulated. Exception represent the mismatch, while match cases are those rows are identical and must even account for jumbled order.

Equivalence Class – Partition	**Input Value-Type**	**Expected Outcome**
Only one value in the array lists is **different** such as the customer id	Boundary Value Csv1: [**ID1**, BOS963211, USD, SAVINGS, 962510] Csv2: [**ID2**, BOS963211, USD, SAVINGS, 962510]	Mismatch-Exception
All rows are different in both arrays	Middle Value Csv1: [ID1, BOS9630233, USD, CURRENT, 932510] Csv2: [ID2, BOS963211, SGD, SAVINGS, 962510]	Mismatch- Exception
All values are same, but array list order is jumbled up in the two arrays.	Boundary Value Csv1: [ID1, BOS963211, **USD, SAVINGS**, 962510] Csv2: [ID2, BOS963211, **SAVINGS, USD**, 962510]	Match
All values are exactly same and so is their order	Middle Value Csv1: [**ID1**, BOS963211, USD, SAVINGS, 962510] Csv2: [**ID1**, BOS963211, USD, SAVINGS, 962510]	Match