I have created a total of 4 classes for this project:
1) "Main" class
2) "MultiHashingTable" class - contains the implementation of Multi Hashing Table.

Important methods:
insert: This inserts a flow Id in the hash table if an empty index is found using the hash generated. If no empty index is found based on the hash generated for inserting the incoming flow id, the flow id is rejected.

3) "DLeftHashTable" class - contains the implementation of the D-Left Hash Table.

Important methods:
insert: This inserts a flow Id in the hash table in the leftmost empty index if found in the hashtable. If no empty index is found using the hash function, the flow id is not inserted into the hashtable.

4) "CuckooHashTable" class - contains the implementation of the Cuckoo Hash table.

Important methods:
insert: inserts a flow id in the hashtable if an empty index is found.

move: if the insert function cannot find an empty index to insert the current flow id in the hashtable, the move function is used to move the existing entries of the hashtable to some other empty location to create space for the incoming flow id if possible. Here, we control the depth of the "recursive" move function by the number of cuckoo steps.

The "Main" class has the "main" function which is the entry point of the program. This main function gives you a list of options and asks from the user for the input.

The list of options displayed are:

1: MultiHash 2: DLeft 3: Cuckoo HashTable 4: exit

Based on the user input, it invokes the implementation methods of MultiHashingTable, DLeftHashTable, and CuckooHashTable class respectively.

Steps to run the program:

1) copy all the .java code files in one folder

2) compile all the .java code files using this command on the command line: javac *.java

3) run the command "java Main" on the command line to invoke the "main" method

of the “Main class”. A list of options will be displayed as:

Enter 1: MultiHash 2: DLeft 3: Cuckoo HashTable 4: exit"

4) select one of the options (1, 2, or 3) from the above options based on which Hash Table code you want to run by entering a number(1,2 or 3) on the command line and press enter. Select 4 if you want to exit the program.

5) The code will run for the type of Hash table selected and output will be generated on the console/terminal as well as the output files will be written in the directory from where the program is being executed.

5) After the console output is generated, the main menu will appear again with the same options as shown in step 3 above so that we can run the program again and again if required.

Currently, all the three Hash Table implementation is called from the "main" java function in the "Main" class using the parameter values given in the project requirements file.

For generating the flow Ids in all the three Hash table implementations, I am using Math.random() function. My flow Ids lie between 1 and Integer.MAX_VALUE in Java.

Also, for the k hash functions, I have defined an integer array of size k as h[k] and initialized with randomly generated values. I am using this to compute Hi(f). So, Hi(f) is computed using H(f XOR h[i]). This is the same logic to implement multiple hash functions as written in the project document shared on canvas by the professor.

HashTable Invocation and Function Parameters:

MultiHashingTable.callMultiHash(1000, 1000, 3) - calls the Multi Hash Table implementation and results are printed on the console as well as an output file is generated. The first parameter in the "callMultiHash" method call is the number of table entries (1000 here), the second parameter is the number of flows (1000 here) and the third parameter is the number of hashes (3 here). If you want to change any of these parameters, change them here in the "main" method of the Main class while calling the "callMultiHash" method of MultiHashingTable class.

DLeftHashTable.callDLeft(1000, 1000, 4) - calls the D-Left Hash Table implementation. The first parameter in the "callDLeft" method is the number of table entries (1000 here), the second parameter is the number of flows (1000 here) and the third parameter is the number of segments (4 here). If you want to change any of these parameters, change them here in the "main" method of the Main class in the "callDLeft" function call of DLeftHashTable class.

CuckooHashTable.callCuckoo(1000, 1000, 3, 2) - Calls the Cuckoo Hash Table implementation and results are printed on the console.

The first parameter in the "callCuckoo" method is the number of table entries (1000 here), the second parameter is the number of flows (1000 here) and the third parameter is the number of hashes (3 here) and 4th parameter is the number of cuckoo steps (2 here). If you want to change any of these parameters, change them here in the "main" method of the "Main" class in the "callCuckoo" function call of CuckooHashTable class.

Output of Program and Output files:

For each of the hash table implementation, the outputs are printed on the console in the below format:
In the first line, the total number of flows in the hashtable is printed. In the next subsequent new lines, the table entries are printed (flow id is printed if the entry has a flow otherwise 0 will be printed).

Also, output files for different hash tables will be created in the current directory every time we run the code for different hash tables by selecting options (1, 2 or 3) from the menu. The file names of different output files are:

1) MultiHashingOutput.txt - output file for the Multi Hashing Table.
2) DLeftOutput.txt - output file for D-Left Hash Table.
3) CuckooHashOutput.txt - output file for Cuckoo Hash Table.

One output file with names as specified above for each hash table is also submitted along with the code files.

The format of the output file is the same as specified in the project description as:

The first line in the output file contains the number of flows in the table. The subsequent lines contain the table entries.