ASSIGNMENT 3

CSCI 6704 – Advanced Topics in Networks
Exercise 1

Dhrumil Amish Shah (B00857606)
dh416386@dal.ca

Exercise 1: In this exercise, you will write a simple program to simulate the bridge-processing flowchart for a bridge that we discussed in the lectures.

Answer:

The source code is developed using IntelliJ IDE and JAVA programming language. The source code consists of the following files inside the **src/main/java** folder:

1. **BridgeProcessingSimulation.java** – This JAVA file contains the core logic for the program execution. The public and private methods inside this file are as below:
    a. Public methods
        i. execute()
    b. Private methods
        i. readFDBDataInMemory()
        ii. performBridgeProcessingSimulation()
        iii. storeOutput()
        iv. storeFDBDataInFile()
2. **BridgeProcessingSimulationTest.java** – This JAVA file contains the main method which executes the BridgeProcessingSimulation class. It contains only one public method (i.e., the main() method)
3. **files** folder – The **files** folder consists of two files used by the BridgeProcessingSimulation class. The names of these files are as below:
    a. BridgeFDB.txt
    b. RandomFrames.txt

BridgeProcessingSimulation class Psuedocode

Step 1: Start

Step 2: The {@code execute()} method of {@code BridgeProcessingSimulation} class is first called from the {@code BridgeProcessingSimulationTest} class.

Step 3: Read the content from BridgeFDB.txt file in memory (i.e. in {@code Map<String, String>)} where {@code key} is MAC address of source and {@code value} is port number using method {@code readFDBDataInMemory()}.

Step 4: Read frames one by one from file RandomFrames.txt and store the source address in {@code sourceAddress}, destination address in {@code destinationAddress} and port in {@code port} using the {@code readLine()} method.

Step 5: Print the current frame under process.

Step 6: Perform the bridge-processing algorithm using method {@code performBridgeProcessingSimulation()} on the current frame under process and return the output which can be {@code "Discarded"}, {@code "Broadcast"}, or {@code "Forwarded on port q"}.

Step 7: Store the output in file BridgeOutput.txt file using method {@code storeOutput()}.

Step 8: After all the frames are processed, store the updated Bridge FDB database from memory to file using {@code storeFDBDataInFile()} method.

Step 9: Stop

Screenshot of successful program execution using file <u>RandomFrames.txt</u>

Figure 1 displays the screenshot of successful program execution using RandomFrames.txt file. The final output is in the file BridgeOutput.txt file and the updated FDB database is in file BridgeFDB.txt.

All assignment files related to this sample run are in below folders:

- b. Text file containing the output for exercise 1 (BridgeOutput.txt file)
- c. BridgeFDB.txt file updated for exercise 1 (BridgeFDB.txt file)

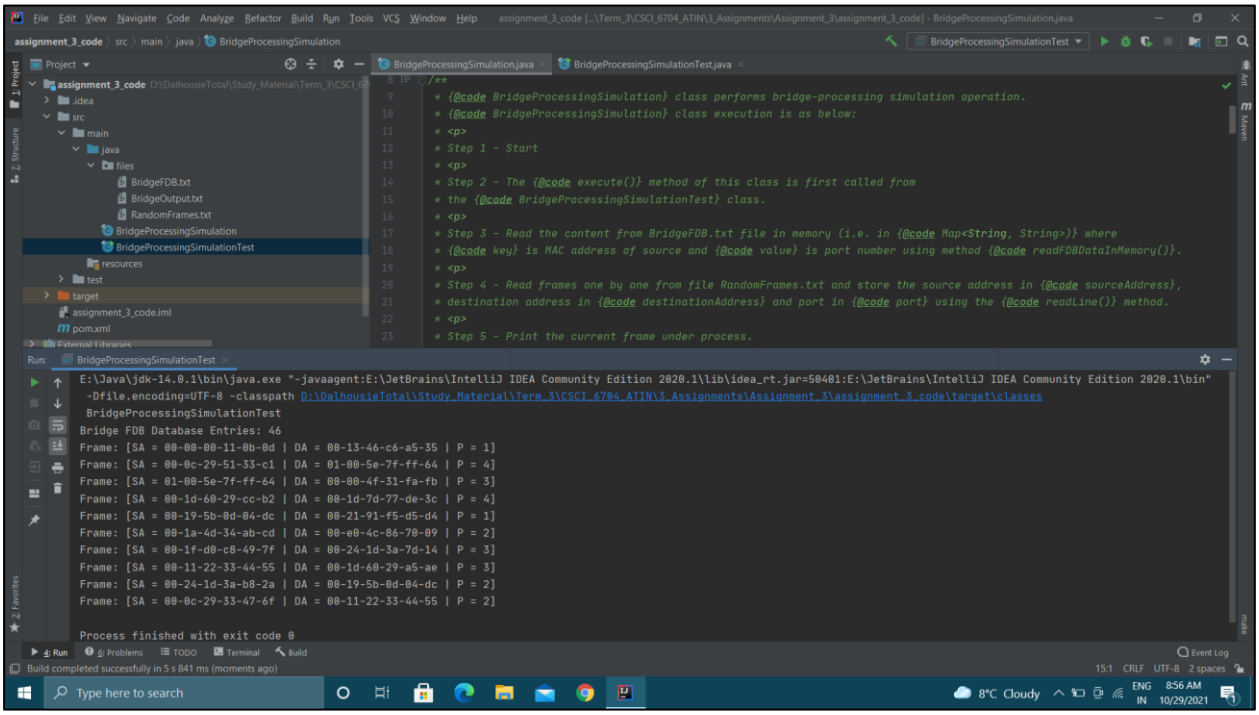

Figure 1 - Screenshot of successful program execution for file RandomFrames.txt

Screenshot of successful program execution using file <u>RandomFramesNew.txt</u>

Figure 2 displays the screenshot of successful program execution using RandomFramesNew.txt file. The final output is in the file BridgeOutput.txt file and the updated FDB database is in file BridgeFDB.txt.

All assignment files related to this sample run are in below folder:

- e. Exercise 1 output for RandomFramesNew.txt file

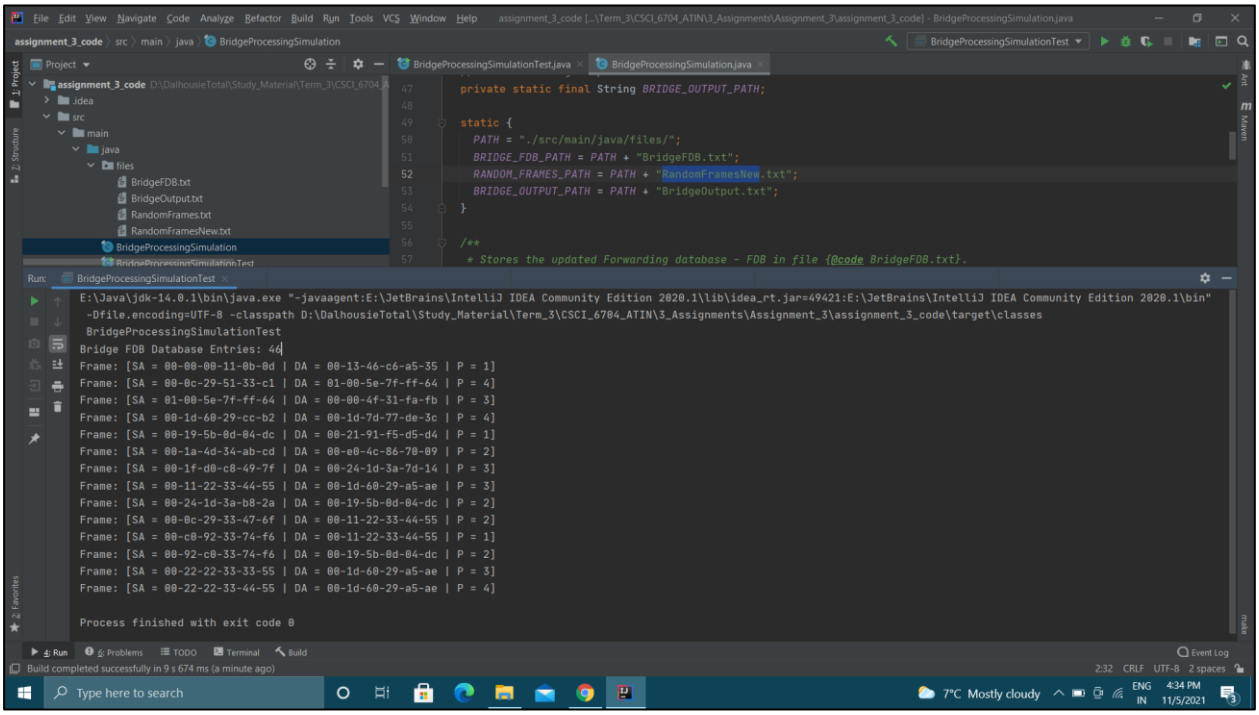

Figure 2 - Screenshot of successful program execution for file RandomFramesNew.txt

Source code folder

The final source code in the below folder:

- a. Source code for exercise 1