

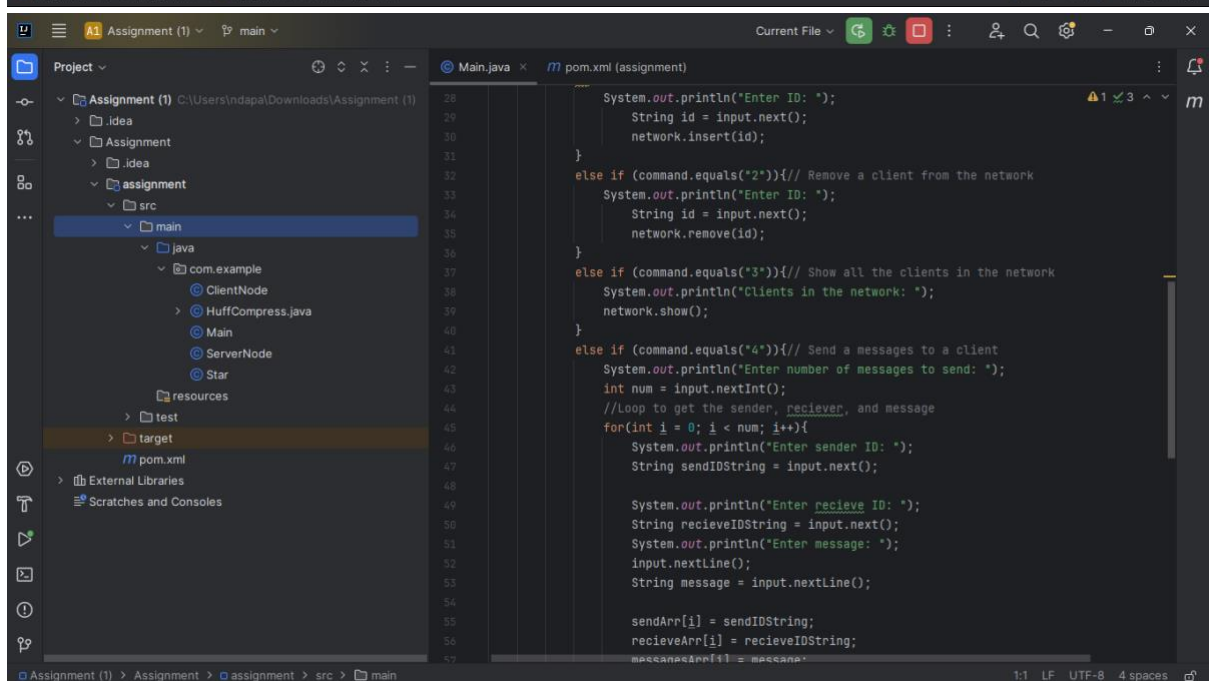
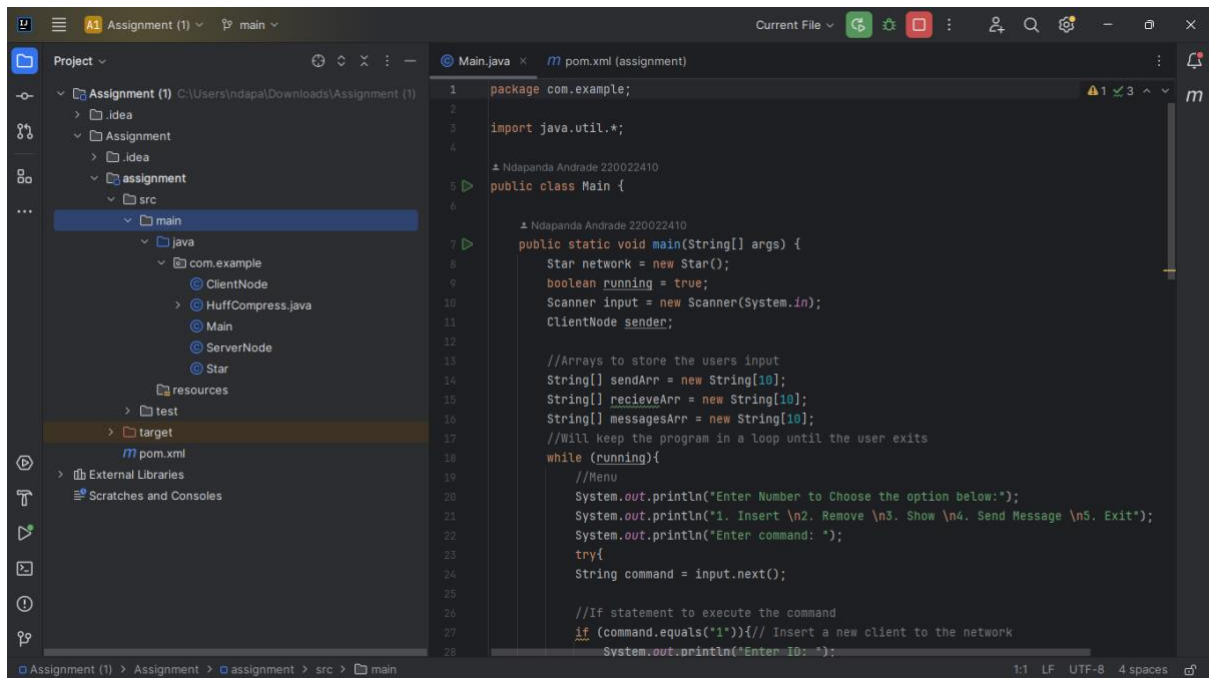
DATA STRUCTURE AND ALGORITHM

GROUP ASSIGNMENT

PT

NANES	STUDENT NO
Nelumbu Paskaris	221061010
Ndapanda Andrade	220022410
David Ndantsi	219068119
kufanga Gerald	22046465

Main



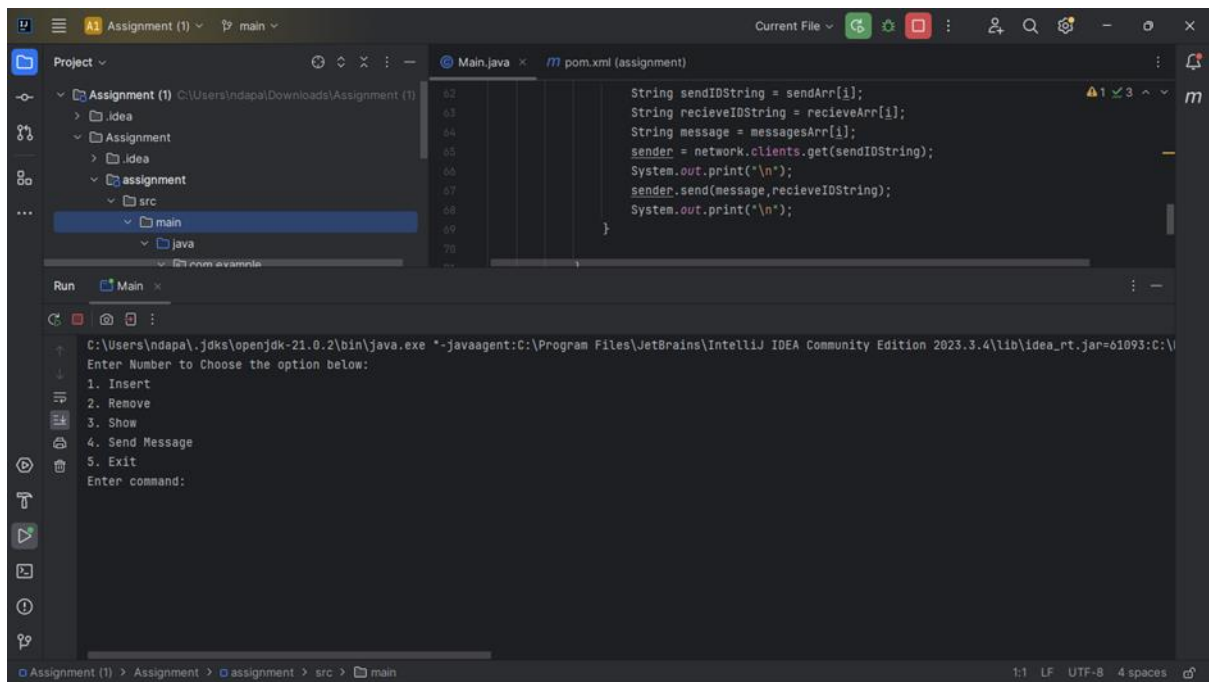
The screenshot shows an IDE window with a project named "Assignment (1)". The project structure on the left includes a "main" directory under "src", which contains a "java" directory. Inside "java", there is a "com.example" package containing "ClientNode", "HuffCompress.java", "Main", "ServerNode", and "Star". There is also a "resources" directory and a "test" directory. The "pom.xml" file is located in the "target" directory. The code editor on the right shows the "Main.java" file. The code includes a loop for sending messages to clients, a command handling section, and an exception catch block.

```
54  
55  
56  
57  
58  
59  
60  
61 //Loop to send the messages to the clients  
62 for(int i = 0; i< num ; i++){  
63     String sendIDString = sendArr[i];  
64     String recieveIDString = recieveArr[i];  
65     String message = messagesArr[i];  
66     sender = network.clients.get(sendIDString);  
67     System.out.print("\n");  
68     sender.send(message,recieveIDString);  
69     System.out.print("\n");  
70 }  
71  
72  
73 else if (command.equals("5")){// Exit the program  
74     running = false;  
75 }  
76 else {  
77     System.out.println("Invalid Command");// If the command is invalid, print error m  
78 }  
79  
80 catch (Exception e){  
81     System.out.println("Invalid Input");  
82 }  
83 }
```

The screenshot shows the same IDE window as the first one, but with a different view of the project structure. The "main" directory is now expanded, showing the "java" directory. Inside "java", there is a "com.example" package containing "ClientNode", "HuffCompress.java", "Main", "ServerNode", and "Star". There is also a "resources" directory and a "test" directory. The "pom.xml" file is located in the "target" directory. The code editor on the right shows the "Main.java" file. The code includes a loop for sending messages to clients, a command handling section, and an exception catch block.

```
58  
59  
60  
61 //Loop to send the messages to the clients  
62 for(int i = 0; i< num ; i++){  
63     String sendIDString = sendArr[i];  
64     String recieveIDString = recieveArr[i];  
65     String message = messagesArr[i];  
66     sender = network.clients.get(sendIDString);  
67     System.out.print("\n");  
68     sender.send(message,recieveIDString);  
69     System.out.print("\n");  
70 }  
71  
72  
73 else if (command.equals("5")){// Exit the program  
74     running = false;  
75 }  
76 else {  
77     System.out.println("Invalid Command");// If the command is invalid, print error m  
78 }  
79  
80 catch (Exception e){  
81     System.out.println("Invalid Input");  
82 }  
83 }  
84  
85 }
```

OUTPUT



The screenshot shows the IntelliJ IDEA IDE with a project named 'Assignment (1)'. The project structure on the left includes a 'main' directory under 'src'. The 'Main.java' file is open, showing the following code:

```
62 String sendIDString = sendArr[i];
63 String recieveIDString = recieveArr[i];
64 String message = messagesArr[i];
65 sender = network.clients.get(sendIDString);
66 System.out.print("\n");
67 sender.send(message, recieveIDString);
68 System.out.print("\n");
69 }
70
```

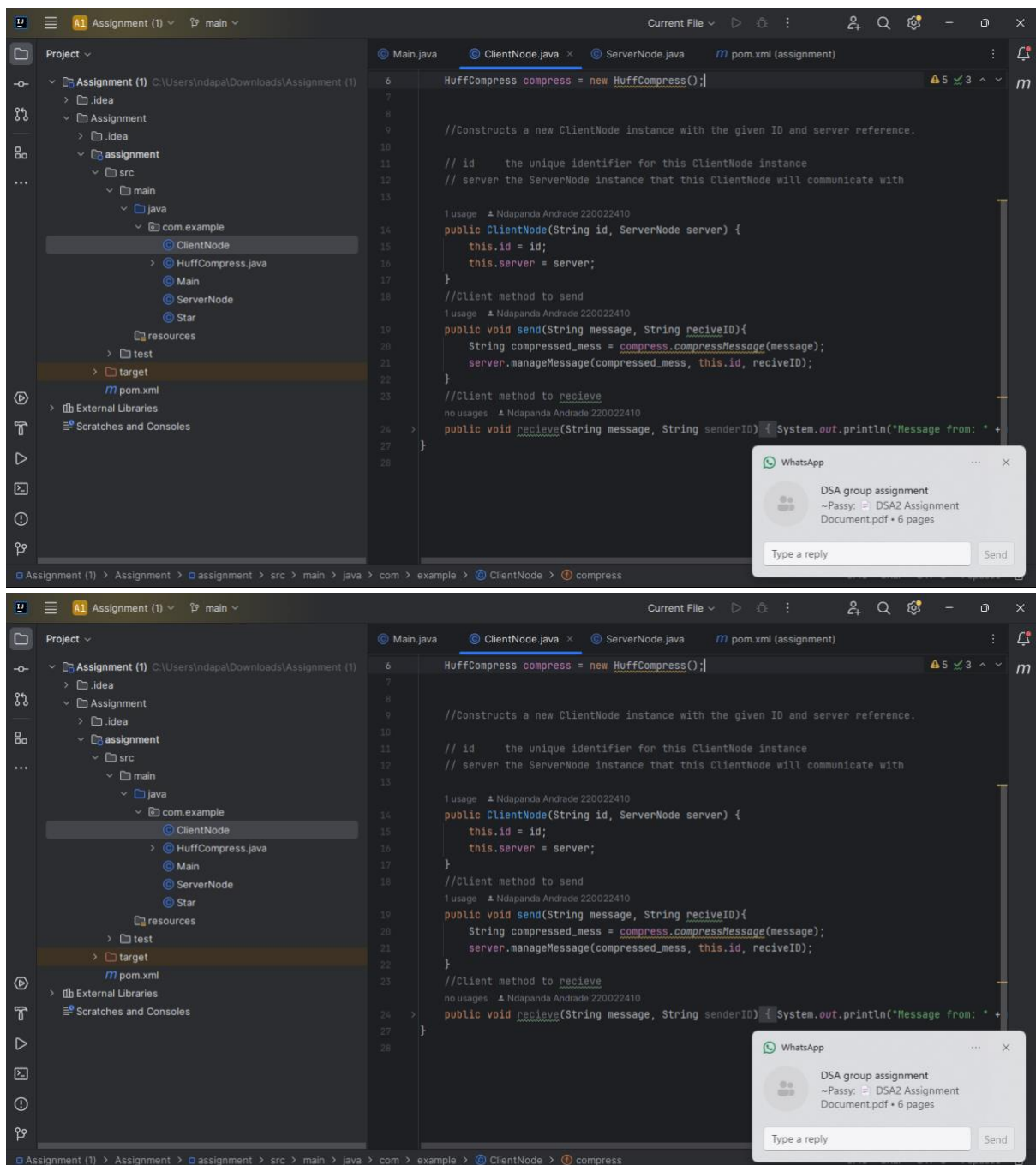
The 'Run' tab at the bottom shows the execution output:

```
C:\Users\ndapa\.jdk\openjdk-21.0.2\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2023.3.4\lib\idea_rt.jar=61093:C:\V
Enter Number to Choose the option below:
1. Insert
2. Remove
3. Show
4. Send Message
5. Exit
Enter command:
```

The status bar at the bottom indicates the file is at line 1, column 1, using UTF-8 encoding with 4 spaces.

JAVA CODE

ClientNode



package com.example;

import java.util.*;

public class Main {

public static void main(String[] args) {

Star network = new Star();

boolean running = true;

Scanner input = new Scanner(System.in);

```

ClientNode sender;

//Arrays to store the users input
String[] sendArr = new String[10];
String[] recieveArr = new String[10];
String[] messagesArr = new String[10];
//Will keep the program in a loop until the user exits
while (running){
    //Menu
    System.out.println("Enter Number to Choose the option below:");
    System.out.println("1. Insert \n2. Remove \n3. Show \n4. Send Message \n5.
Exit");
    System.out.println("Enter command: ");
    try{
        String command = input.next();

        //If statement to execute the command
        if (command.equals("1")){// Insert a new client to the network
            System.out.println("Enter ID: ");
            String id = input.next();
            network.insert(id);
        }
        else if (command.equals("2")){// Remove a client from the network
            System.out.println("Enter ID: ");
            String id = input.next();
            network.remove(id);
        }
        else if (command.equals("3")){// Show all the clients in the network
            System.out.println("Clients in the network: ");
            network.show();
        }
        else if (command.equals("4")){// Send a messages to a client
            System.out.println("Enter number of messages to send: ");
            int num = input.nextInt();
            //Loop to get the sender, reciever, and message
            for(int i = 0; i < num; i++){
                System.out.println("Enter sender ID: ");
                String sendIDString = input.next();

                System.out.println("Enter recieve ID: ");
                String recieveIDString = input.next();
            }
        }
    }
}

```

```

        System.out.println("Enter message: ");
        input.nextLine();
        String message = input.nextLine();

        sendArr[i] = sendIDString;
        recieveArr[i] = recieveIDString;
        messagesArr[i] = message;

    }
    //Loop to send the messages to the clients
    for(int i = 0; i< num ; i++){
        String sendIDString = sendArr[i];
        String recieveIDString = recieveArr[i];
        String message = messagesArr[i];
        sender = network.clients.get(sendIDString);
        System.out.print("\n");
        sender.send(message,recieveIDString);
        System.out.print("\n");
    }

}

else if (command.equals("5")){// Exit the program
    running = false;
}
else {
    System.out.println("Invalid Command");// If the command is invalid, print error
message
    }
}
catch (Exception e){
    System.out.println("Invalid Input");
}
}
}
}

```

SeverNodeClass

```
package com.example;
```

```
public class ClientNode {
```

```
    String id;
```

```
    ServerNode server;
```

```
    HuffCompress compress = new HuffCompress();
```

```
    //Constructs a new ClientNode instance with the given ID and server reference.
```

```
    // id    the unique identifier for this ClientNode instance
```

```
    // server the ServerNode instance that this ClientNode will communicate with
```

```
    public ClientNode(String id, ServerNode server) {
```

```
        this.id = id;
```

```
        this.server = server;
```

```
    }
```

```
    //Client method to send
```

```
    public void send(String message, String receiveID){
```

```
        String compressed_mess = compress.compressMessage(message);
```

```
        server.manageMessage(compressed_mess, this.id, receiveID);
```

```
    }
```

```
    //Client method to receive
```

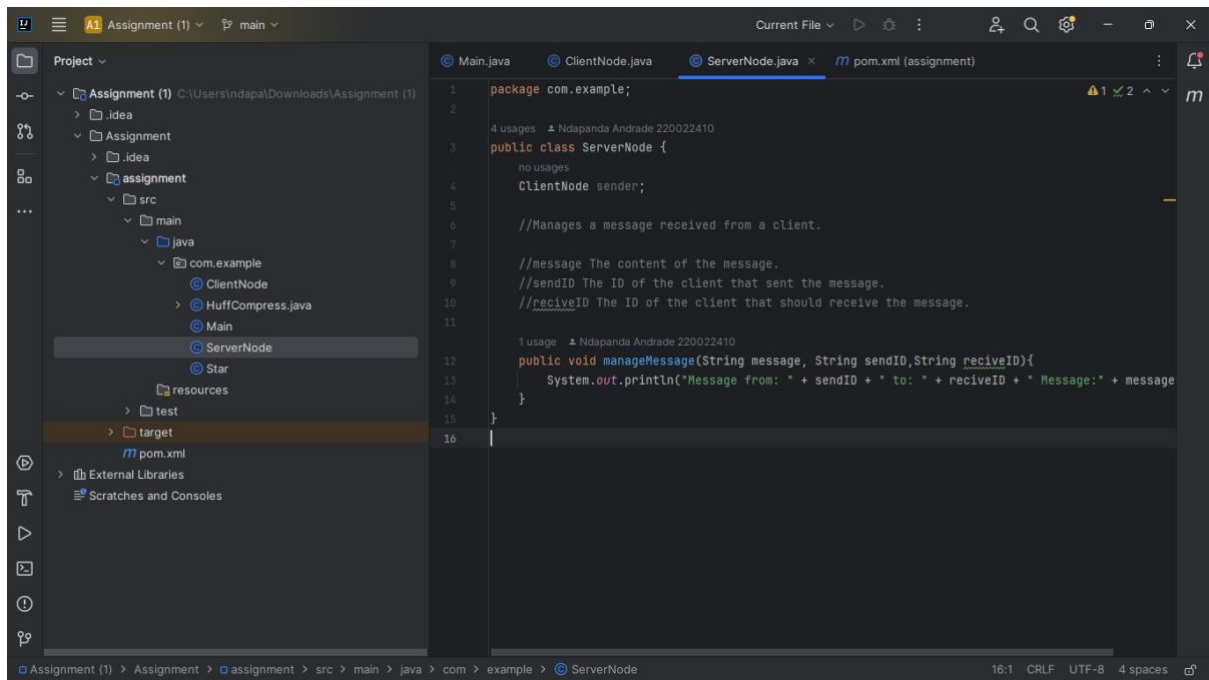
```
    public void receive(String message, String senderID){
```

```
        System.out.println("Message from: " + message + "");
```

```
    }
```

```
}
```

```
ServerNode
```

```
package com.example;
```

```
public class ServerNode {
    ClientNode sender;
```

```
    //Manages a message received from a client.
```

```
    //message The content of the message.
```

```
    //sendID The ID of the client that sent the message.
```

```
    //receiveID The ID of the client that should receive the message.
```

```
    public void manageMessage(String message, String sendID, String receiveID) {
        System.out.println("Message from: " + sendID + " to: " + receiveID + " Message:" +
message);
    }
}
```

Star

```
package com.example;
```

```
import java.util.*;
```

```
public class Star {
```

```
    ServerNode server;
```

```
    Map<String, ClientNode> clients = new HashMap<>();
```

```

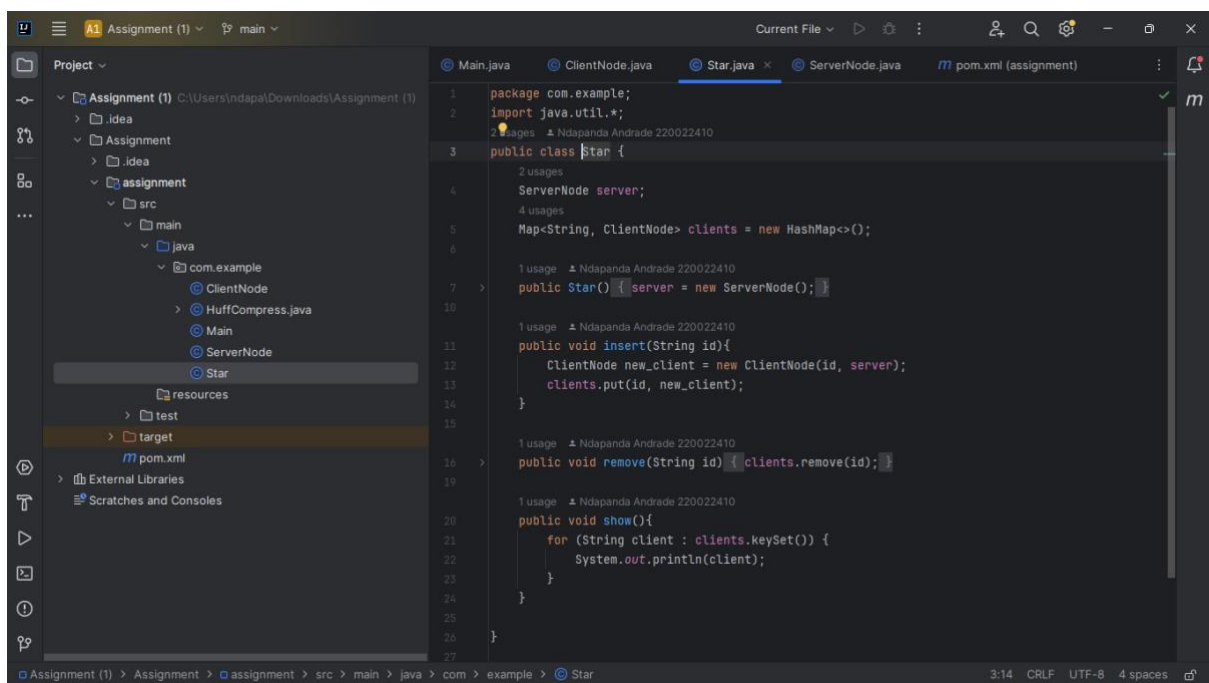
public Star() {
    server = new ServerNode();
}

public void insert(String id){
    ClientNode new_client = new ClientNode(id, server);
    clients.put(id, new_client);
}

public void remove(String id){
    clients.remove(id);
}

public void show(){
    for (String client : clients.keySet()) {
        System.out.println(client);
    }
}
}

```



Data Compression

Compression

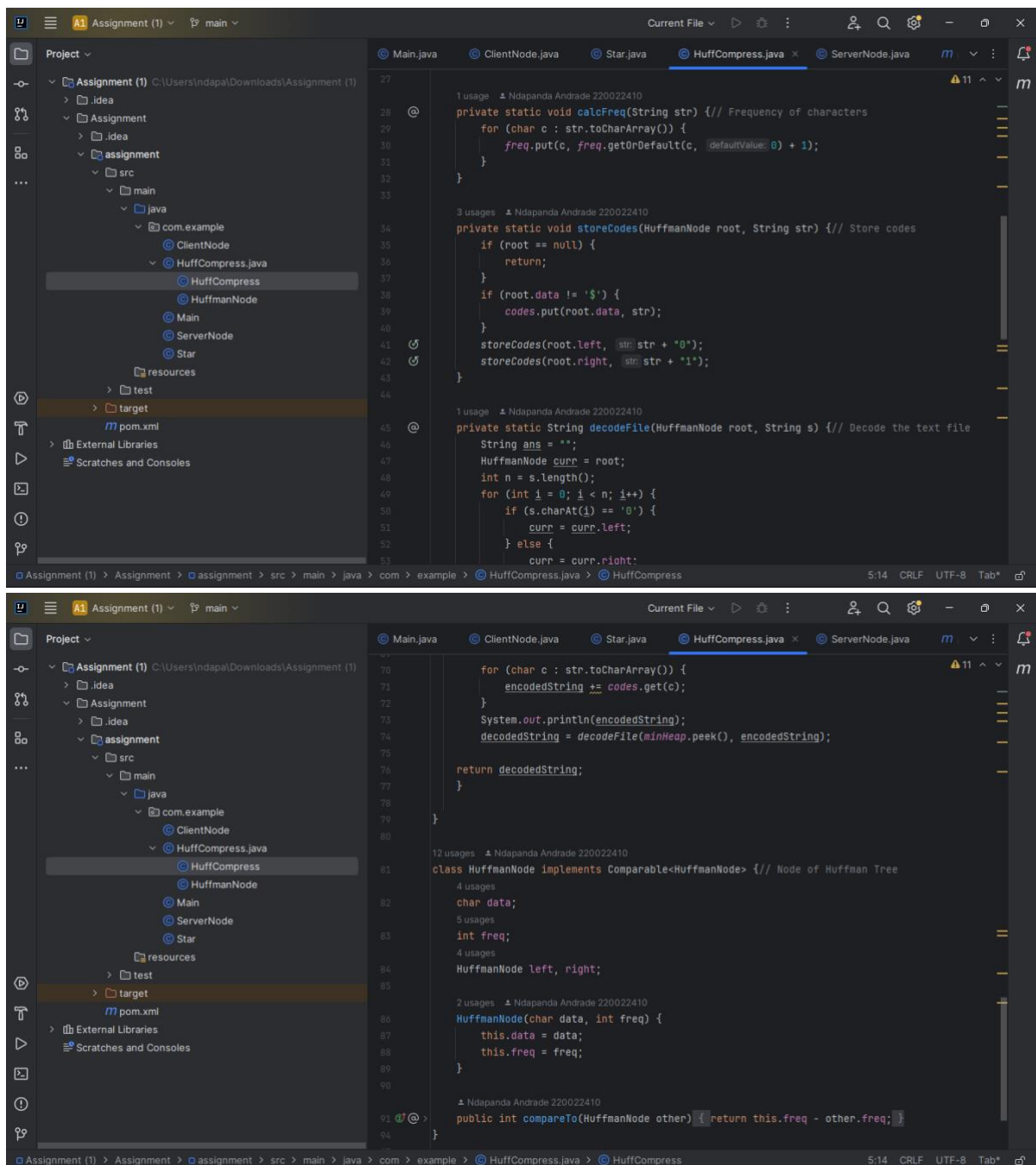
The image displays two screenshots of an IDE (IntelliJ IDEA) showing the implementation of a Huffman compression algorithm in Java. The project structure on the left indicates a package `com.example` with classes `ClientNode`, `HuffCompress`, `HuffmanNode`, `Main`, `ServerNode`, and `Star`.

Top Screenshot: HuffmanCodes Method

```
1 package com.example;
2 import java.util.*;
3
4 2 usages Ndapanda Andrade 220022410
5 public class HuffCompress {
6
7 2 usages
8 private static Map<Character, String> codes = new HashMap<>();
9 3 usages
10 private static Map<Character, Integer> freq = new HashMap<>();
11 7 usages
12 private static PriorityQueue<HuffmanNode> minHeap = new PriorityQueue<>();
13
14 1 usage Ndapanda Andrade 220022410
15 private static void HuffmanCodes(int size) { // Huffman Codes
16     for (Entry<Character, Integer> entry : freq.entrySet()) {
17         minHeap.add(new HuffmanNode(entry.getKey(), entry.getValue()));
18     }
19     while (minHeap.size() != 1) {
20         HuffmanNode left = minHeap.poll();
21         HuffmanNode right = minHeap.poll();
22         HuffmanNode top = new HuffmanNode(data: '$', freq: left.freq + right.freq);
23         top.left = left;
24         top.right = right;
25         minHeap.add(top);
26     }
27     storeCodes(minHeap.peek(), str: "");
28 }
```

Bottom Screenshot: calcFreq, storeCodes, and decodeFile Methods

```
25 storeCodes(minHeap.peek(), str: "");
26 }
27
28 1 usage Ndapanda Andrade 220022410
29 @ private static void calcFreq(String str) { // Frequency of characters
30     for (char c : str.toCharArray()) {
31         freq.put(c, freq.getOrDefault(c, defaultValue: 0) + 1);
32     }
33 }
34
35 3 usages Ndapanda Andrade 220022410
36 private static void storeCodes(HuffmanNode root, String str) { // Store codes
37     if (root == null) {
38         return;
39     }
40     if (root.data != '$') {
41         codes.put(root.data, str);
42     }
43     storeCodes(root.left, str: str + '0');
44     storeCodes(root.right, str: str + '1');
45 }
46
47 1 usage Ndapanda Andrade 220022410
48 @ private static String decodeFile(HuffmanNode root, String s) { // Decode the text file
49     String ans = "";
50     HuffmanNode curr = root;
51     int n = s.length();
52     for (int i = 0; i < n; i++) {
53         if (s.charAt(i) == '0') {
54             curr = curr.left;
55         } else {
56             curr = curr.right;
57         }
58         if (curr.data != '$') {
59             ans += curr.data;
60             curr = root;
61         }
62     }
63     return ans;
64 }
```



Java code

```
package com.example;
```

```
import java.util.*;
```

```
import java.util.Map.Entry;
```

```
public class HuffCompress {
```

```
    private static Map<Character, String> codes = new HashMap<>();
```

```
    private static Map<Character, Integer> freq = new HashMap<>();
```

```
    private static PriorityQueue<HuffmanNode> minHeap = new PriorityQueue<>();
```

```

private static void HuffmanCodes(int size) { // Huffman Codes
    for (Entry<Character, Integer> entry : freq.entrySet()) {
        minHeap.add(new HuffmanNode(entry.getKey(), entry.getValue()));
    }
    while (minHeap.size() != 1) {
        HuffmanNode left = minHeap.poll();
        HuffmanNode right = minHeap.poll();
        HuffmanNode top = new HuffmanNode('$', left.freq + right.freq);
        top.left = left;
        top.right = right;
        minHeap.add(top);
    }
    storeCodes(minHeap.peek(), "");
}

```

```

private static void calcFreq(String str) { // Frequency of characters
    for (char c : str.toCharArray()) {
        freq.put(c, freq.getOrDefault(c, 0) + 1);
    }
}

```

```

private static void storeCodes(HuffmanNode root, String str) { // Store codes
    if (root == null) {
        return;
    }
    if (root.data != '$') {
        codes.put(root.data, str);
    }
    storeCodes(root.left, str + "0");
    storeCodes(root.right, str + "1");
}

```

```

private static String decodeFile(HuffmanNode root, String s) { // Decode the text file
    String ans = "";
    HuffmanNode curr = root;
    int n = s.length();
    for (int i = 0; i < n; i++) {
        if (s.charAt(i) == '0') {

```

```

        curr = curr.left;
    } else {
        curr = curr.right;
    }
    if (curr.left == null && curr.right == null) {
        ans += curr.data;
        curr = root;
    }
}
return ans + '\0';
}

```

```

public static String compressMessage(String message) { // Compresses the message
    String str = message;
    String encodedString = "";
    String decodedString = "";
    calcFreq(str);
    HuffmanCodes(str.length());

    for (char c : str.toCharArray()) {
        encodedString += codes.get(c);
    }
    System.out.println(encodedString);
    decodedString = decodeFile(minHeap.peek(), encodedString);

    return decodedString;
}
}

```

```

class HuffmanNode implements Comparable<HuffmanNode> { // Node of Huffman Tree
    char data;
    int freq;
    HuffmanNode left, right;

    HuffmanNode(char data, int freq) {
        this.data = data;
        this.freq = freq;
    }

    public int compareTo(HuffmanNode other) {

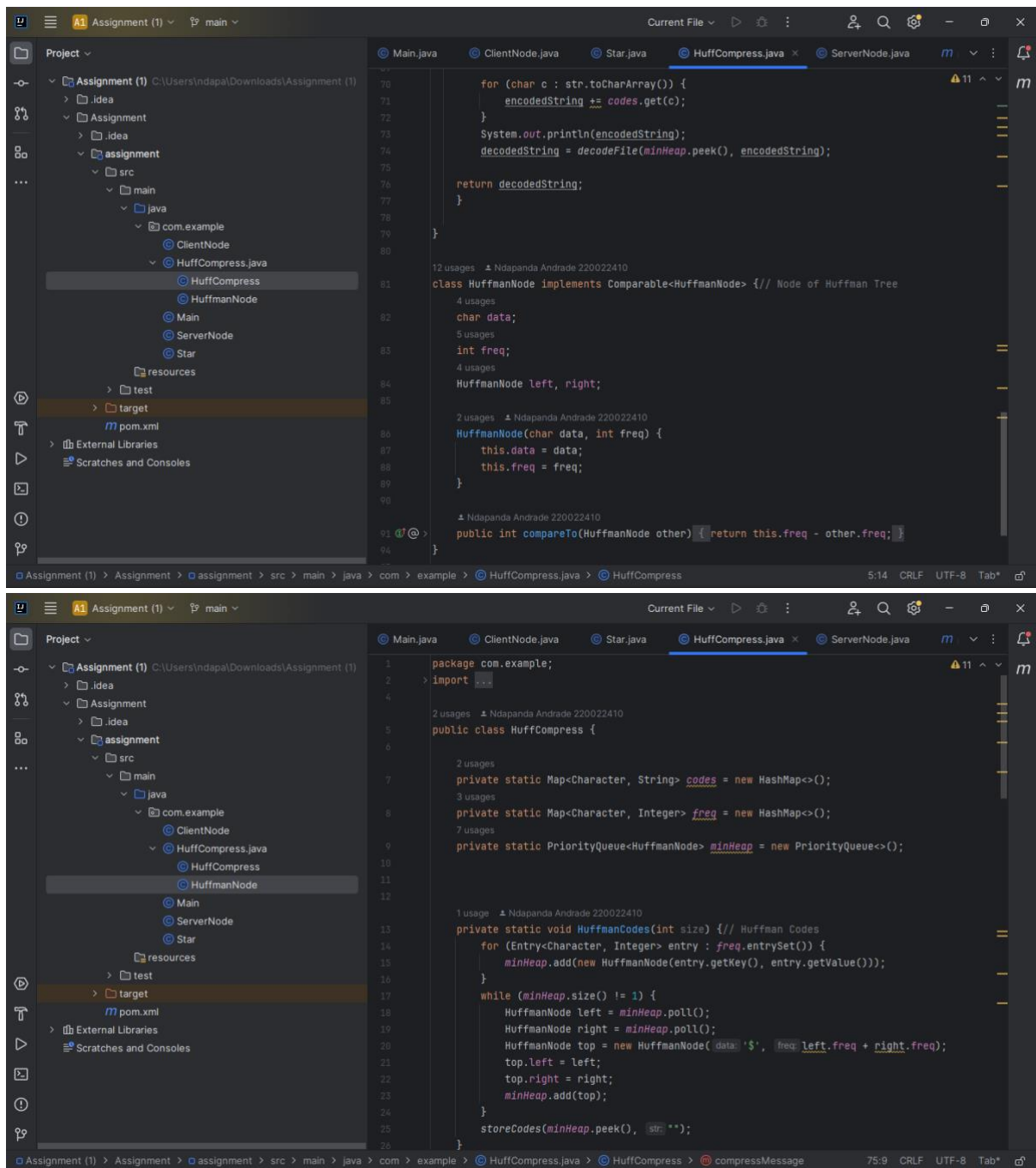
```

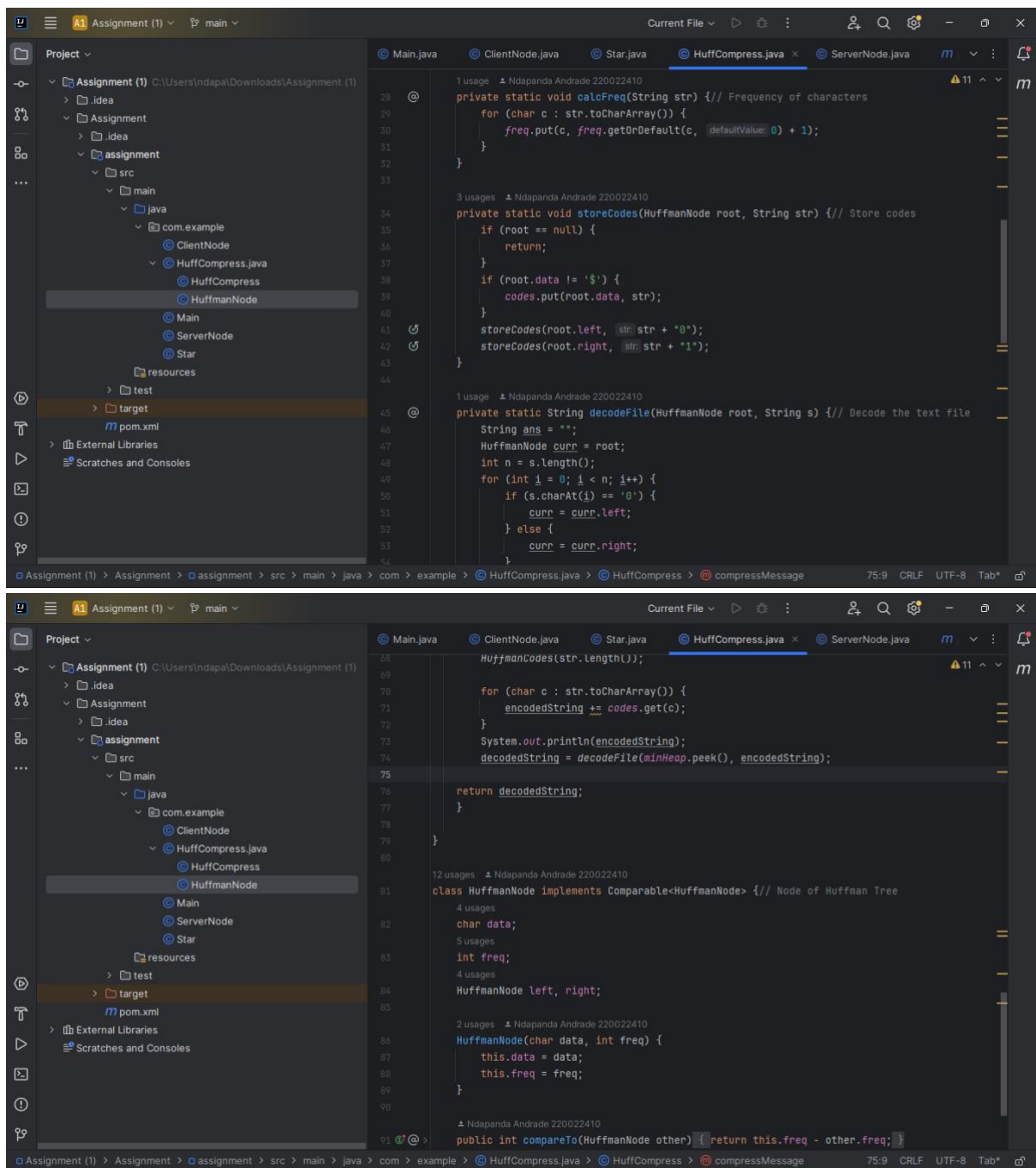
```

    return this.freq - other.freq;
}
}

```

Node





Java code

package com.example;

import java.util.*;

import java.util.Map.Entry;

public class HuffCompress {

private static Map<Character, String> codes = new HashMap<>();


```
private static Map<Character, Integer> freq = new HashMap<>();  
private static PriorityQueue<HuffmanNode> minHeap = new PriorityQueue<>();
```

```
private static void HuffmanCodes(int size) { // Huffman Codes  
    for (Entry<Character, Integer> entry : freq.entrySet()) {  
        minHeap.add(new HuffmanNode(entry.getKey(), entry.getValue()));  
    }  
    while (minHeap.size() != 1) {  
        HuffmanNode left = minHeap.poll();  
        HuffmanNode right = minHeap.poll();  
        HuffmanNode top = new HuffmanNode('$', left.freq + right.freq);  
        top.left = left;  
        top.right = right;  
        minHeap.add(top);  
    }  
    storeCodes(minHeap.peek(), "");  
}
```

```
private static void calcFreq(String str) { // Frequency of characters  
    for (char c : str.toCharArray()) {  
        freq.put(c, freq.getOrDefault(c, 0) + 1);  
    }  
}
```

```
private static void storeCodes(HuffmanNode root, String str) { // Store codes  
    if (root == null) {  
        return;  
    }  
    if (root.data != '$') {  
        codes.put(root.data, str);  
    }  
    storeCodes(root.left, str + "0");  
    storeCodes(root.right, str + "1");  
}
```

```
private static String decodeFile(HuffmanNode root, String s) { // Decode the text file  
    String ans = "";  
    HuffmanNode curr = root;  
    int n = s.length();
```

```

    for (int i = 0; i < n; i++) {
        if (s.charAt(i) == '0') {
            curr = curr.left;
        } else {
            curr = curr.right;
        }
        if (curr.left == null && curr.right == null) {
            ans += curr.data;
            curr = root;
        }
    }
    return ans + '\0';
}

```

```

public static String compressMessage(String message) { // Compresses the message
    String str = message;
    String encodedString = "";
    String decodedString = "";
    calcFreq(str);
    HuffmanCodes(str.length());

    for (char c : str.toCharArray()) {
        encodedString += codes.get(c);
    }
    System.out.println(encodedString);
    decodedString = decodeFile(minHeap.peek(), encodedString);

    return decodedString;
}
}

```

```

class HuffmanNode implements Comparable<HuffmanNode> { // Node of Huffman Tree
    char data;
    int freq;
    HuffmanNode left, right;

    HuffmanNode(char data, int freq) {
        this.data = data;
        this.freq = freq;
    }
}

```

```
public int compareTo(HuffmanNode other) {  
    return this.freq - other.freq;  
}  
}
```