95702: Distributed Systems for Information Systems Management

Project 2

Name: Ariane Correa
Andrew Email ID: ajcorrea@andrew.cmu.edu

Task 0:

Project2Task0Client

EchoClientUDP.java | EchoServerUDP.java

```java
// Ariane Correa
// ajcorrea

package com.example.project2task0;

import java.net.*;
import java.io.*;
import java.util.Scanner;

// Reference: EchoClientUDP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)

public class EchoClientUDP {

    // args[] gives server hostname and message content
    public static void main(String args[]) {

        DatagramSocket aSocket = null; // Declare a DatagramSocket for UDP communication.

        try {
            System.out.println("The UDP client is running.");

            // Define the server host (in this case, localhost).
            InetAddress aHost = InetAddress.getByName( host: "localhost");

            Scanner sc = new Scanner(System.in);
            System.out.println("Provide server side port number");

            int serverPort = sc.nextInt(); // Read the server's port number from the user.

            aSocket = new DatagramSocket(); // Create a DatagramSocket for sending and receiving UDP packets.
            String nextLine;
            BufferedReader typed = new BufferedReader(new InputStreamReader(System.in));

            while ((nextLine = typed.readLine()) != null) {

```

Project2Task0Client (continued)

EchoClientUDP.java × EchoServerUDP.java

```
                // Create a DatagramPacket for sending data to the server.
                DatagramPacket request = new DatagramPacket(m, m.length, aHost, serverPort);
                aSocket.send(request); // Send the data to the server.

                // Initialize a new buffer that will store contents of the reply message
                byte[] buffer = new byte[1000];
                // Create a DatagramPacket for receiving data from the server.
                DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
                aSocket.receive(reply); // Receive data from the server.

                // We create a new replyArray of request length
                byte[] replyArray = new byte[request.getLength()];
                // Copy relevant reply data into the new array based on request length
                System.arraycopy(reply.getData(), srcPos: 0, replyArray, destPos: 0, reply.getLength());
                String requestString = new String(replyArray); // Convert received bytes to a string.
                System.out.println("Reply: " + requestString); // Print the server's reply.

                // Exit if "halt!" is received
                if (requestString.equalsIgnoreCase( anotherString: "halt!")) {
                    System.out.println("UDP Client side quitting");
                    break;
                }
            }
        }
        catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage()); // Handle SocketException if it occurs.
        }
        catch (IOException e) {
            System.out.println("IO: " + e.getMessage()); // Handle IOException if it occurs.
        }
        finally {
            if (aSocket != null) aSocket.close(); // Close the DatagramSocket when done.
        }
    }
}
```

Project2Task0Server

EchoClientUDP.java | EchoServerUDP.java

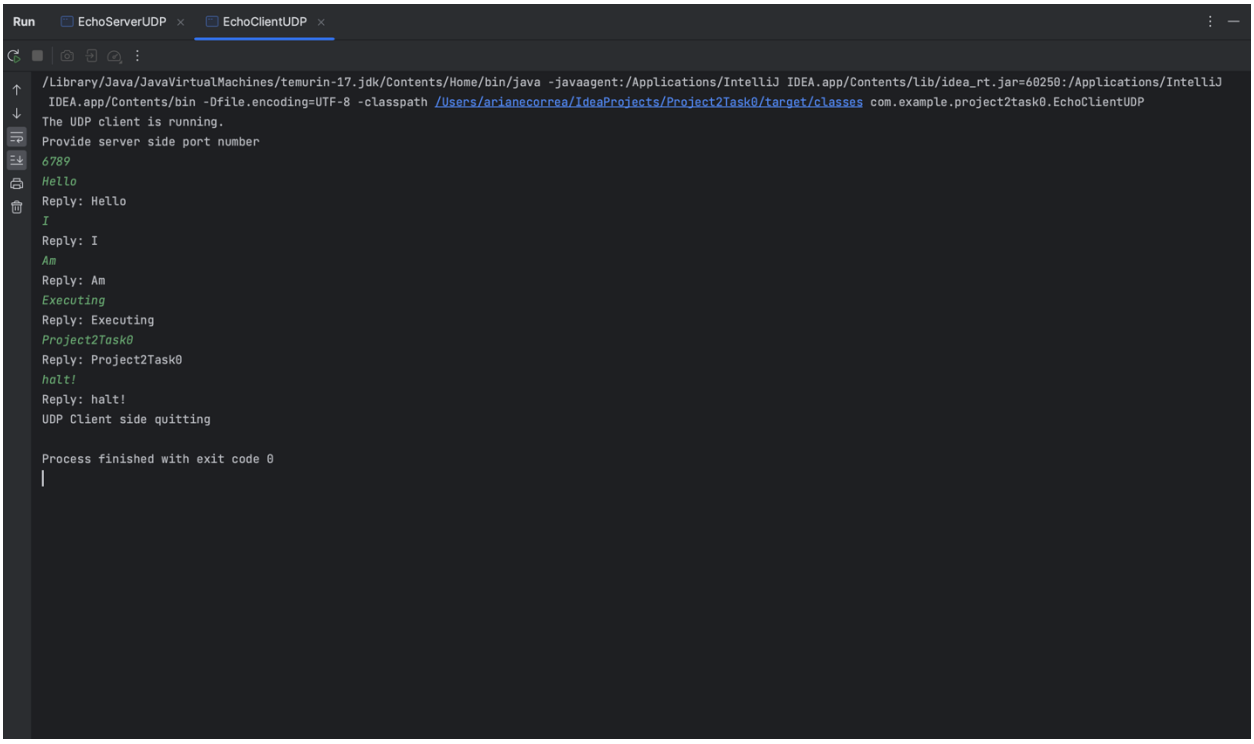
```java
// Ariane Correa
// ajcorrea

package com.example.project2task0;

import java.net.*;
import java.io.*;
import java.util.Arrays;
import java.util.Scanner;

// Reference: EchoServerUDP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)

public class EchoServerUDP {
    public static void main(String args[]) {

        System.out.println("The UDP server is running.");

        DatagramSocket aSocket = null; // Declare a DatagramSocket for UDP communication.

        // We declare a byte array that will receive client messages
        byte[] buffer = new byte[1000];
        try {

            Scanner sc = new Scanner(System.in);
            System.out.println("Provide port number server should listen on");
            int serverPort = sc.nextInt(); // Read the port number from the user.
            aSocket = new DatagramSocket(serverPort); // Create a DatagramSocket bound to the specified port.
            // Create a new datagram packet to fetch the request
            DatagramPacket request = new DatagramPacket(buffer, buffer.length);
            while (true) {

                aSocket.receive(request); // Receive a UDP packet and store it in 'request'.

                byte[] replyArray = new byte[request.getLength()];

```

Project2Task0Server (continued)

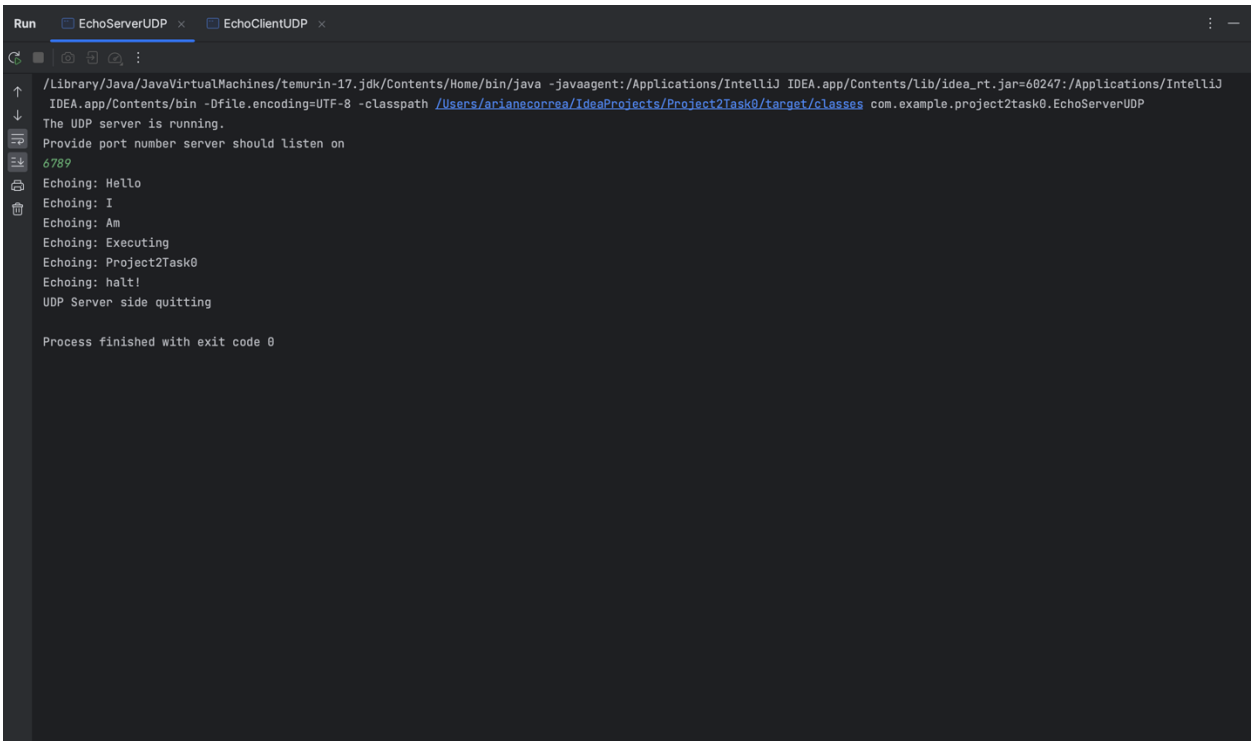
EchoClientUDP.java | EchoServerUDP.java

```
36                  // Copy the message to the reply array which is the length of the request
37                  System.arraycopy(request.getData(), srcPos: 0, replyArray, destPos: 0, request.getLength());
38                  String requestString = new String(replyArray); // Convert the received bytes to a string.
39                  System.out.println("Echoing: " + requestString);
40
41                  // Create a new datagram packet with the newly created reply array, request hosts and port array
42                  DatagramPacket reply = new DatagramPacket(replyArray,
43                          request.getLength(), request.getAddress(), request.getPort());
44
45                  aSocket.send(reply); // Send a reply back to the client.
46
47                  // If the user enters "halt!", Server shuts down
48                  if (requestString.equalsIgnoreCase( anotherString: "halt!")) {
49                      System.out.println("UDP Server side quitting");
50                      break;
51                  }
52              }
53          }
54          catch (SocketException e) {
55              System.out.println("Socket: " + e.getMessage()); // Handle SocketException if it occurs.
56          }
57          catch (IOException e) {
58              System.out.println("IO: " + e.getMessage()); // Handle IOException if it occurs.
59          }
60          finally {
61              if (aSocket != null) aSocket.close(); // Close the DatagramSocket when done.
62          }
63      }
64  }
65
```

Project2Task0ClientConsole



Project2Task0ServerConsole



95702: Distributed Systems for Information Systems Management

Project 2

Name: Ariane Correa
Andrew Email ID: ajcorrea@andrew.cmu.edu

Task 1:

EavesdropperUDP.java

EchoClientUDP.java | EavesdropperUDP.java | EchoServerUDP.java

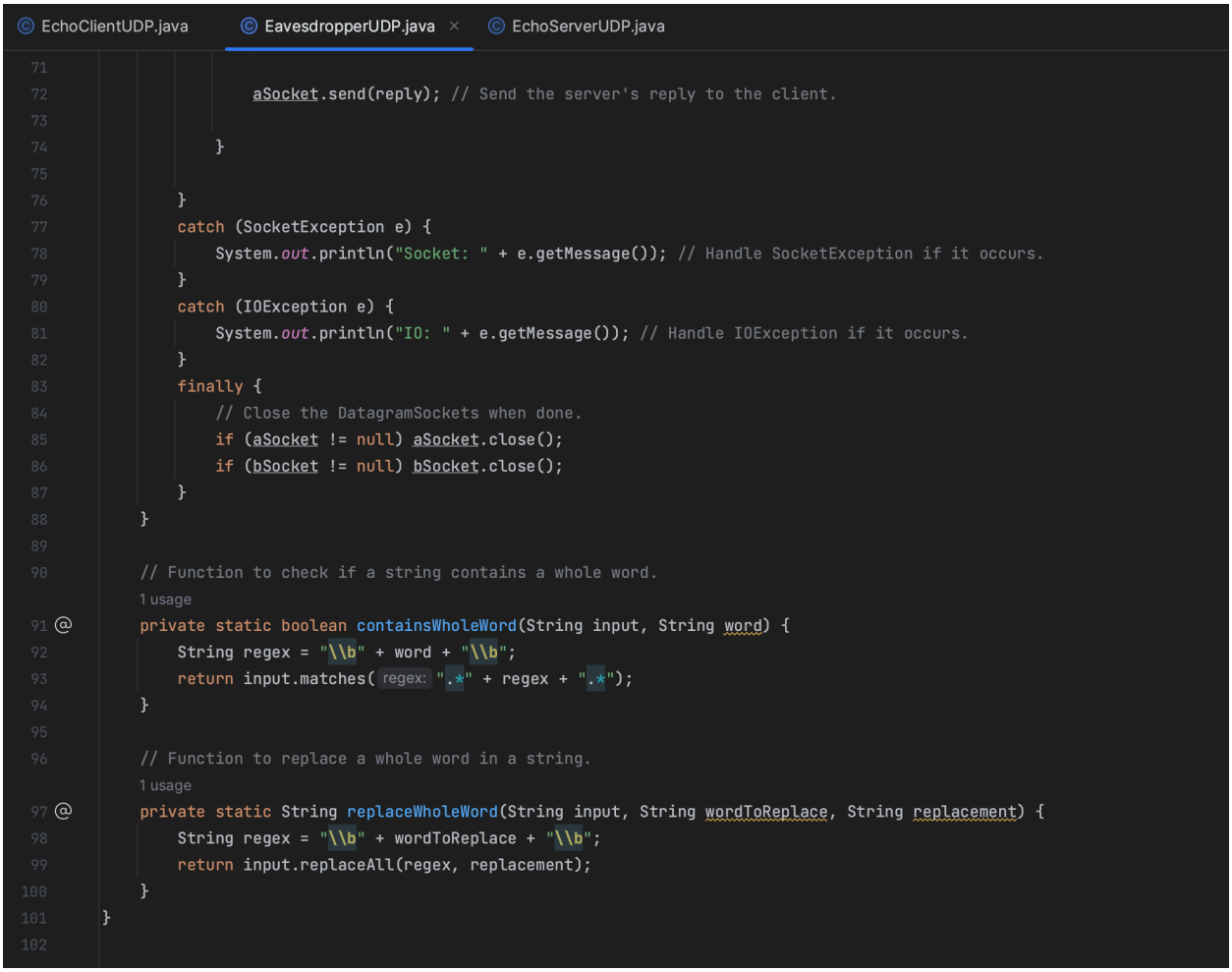
```java
// Ariane Correa
// ajcorrea

package com.example.project2task1;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.util.Scanner;

public class EavesdropperUDP {

    public static void main(String args[]) {

        System.out.println("The Eavesdropper UDP is running.");

        DatagramSocket aSocket = null; // Socket for listening to the client.
        DatagramSocket bSocket = null; // Socket for communication with the server.

        byte[] buffer = new byte[1000];

        try {

            // Define the server host (in this case, localhost).
            InetAddress aHost = InetAddress.getByName( host: "localhost");
            Scanner sc = new Scanner(System.in);
            System.out.println("Provide port number Eavesdropper should listen on");
            int serverPortListener = sc.nextInt(); // Read the port number Eavesdropper should listen on
            System.out.println("Provide port number Eavesdropper is masquerading as");
            int serverPortMasq = sc.nextInt(); // Read the port number Eavesdropper is masquerading as

            aSocket = new DatagramSocket(serverPortListener); // Listen for client messages.
            bSocket = new DatagramSocket(); // Create a socket for communicating with the server.
```

EavesdropperUDP.java (continued)

EchoClientUDP.java | EavesdropperUDP.java | EchoServerUDP.java

```
36
37          DatagramPacket clientRequest = new DatagramPacket(buffer, buffer.length);
38
39          while (true) {
40
41              aSocket.receive(clientRequest); // Receive client's request.
42
43              //copy the relevant message to the reply array , which would be the length of the request
44              byte[] replyArray = new byte[clientRequest.getLength()];
45              System.arraycopy(clientRequest.getData(), srcPos: 0, replyArray, destPos: 0, clientRequest.getLength());
46              String requestString = new String(replyArray);
47              System.out.println("Client Message: " + requestString);
48
49              // Check if the request contains the word "like" as a whole word.
50              if (containsWholeWord(requestString, word: "like")) {
51                  // ACTIVE ATTACK: Replace "like" with "dislike" in the client's request.
52                  requestString = replaceWholeWord(requestString, wordToReplace: "like", replacement: "dislike");
53              }
54
55              byte[] m = requestString.getBytes();
56              DatagramPacket serverRequest = new DatagramPacket(m, m.length, aHost, serverPortMasq);
57              bSocket.send(serverRequest); // Forward the modified request to the server.
58              bSocket.receive(serverRequest); // Receive server's reply.
59
60              byte[] serverReplyArray = new byte[serverRequest.getLength()];
61              System.arraycopy(serverRequest.getData(), srcPos: 0,
62                      serverReplyArray, destPos: 0, serverRequest.getLength());
63              String serverReply = new String(serverReplyArray);
64
65              System.out.println("Server Reply: " + serverReply);
66
67              byte[] n = serverReply.getBytes();
68
69              DatagramPacket reply = new DatagramPacket(n,
70                      n.length, clientRequest.getAddress(), clientRequest.getPort());
```

EavesdropperUDP.java (continued)

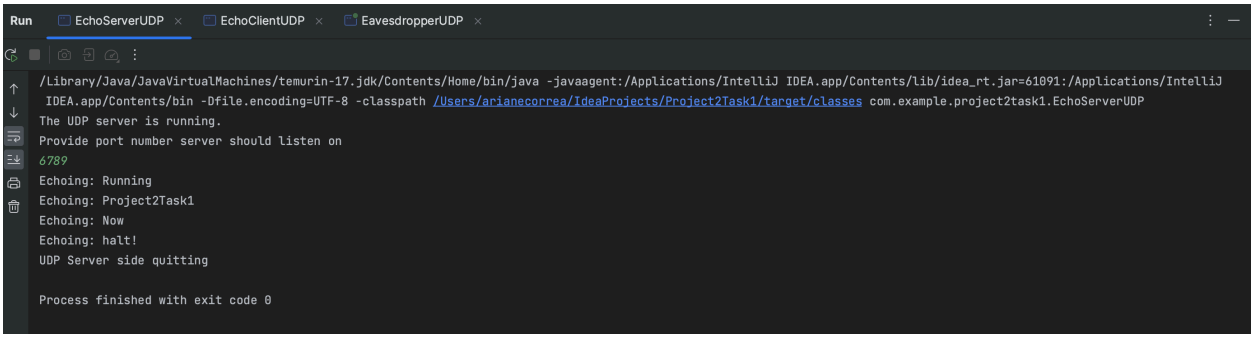


Project2Task1ThreeConsoles

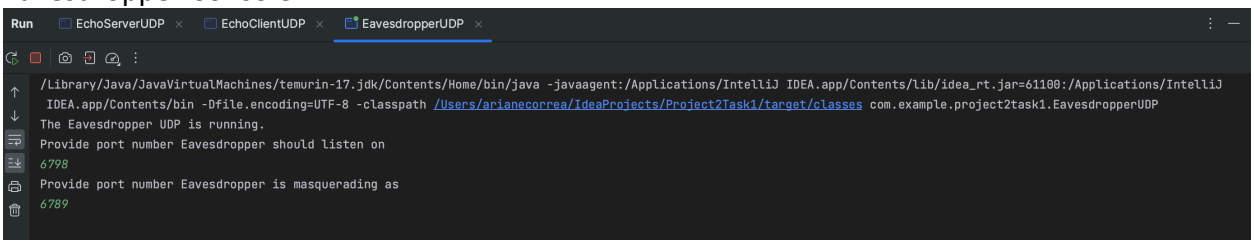
Case 1: *Correct Server*

Client Console

Server Console



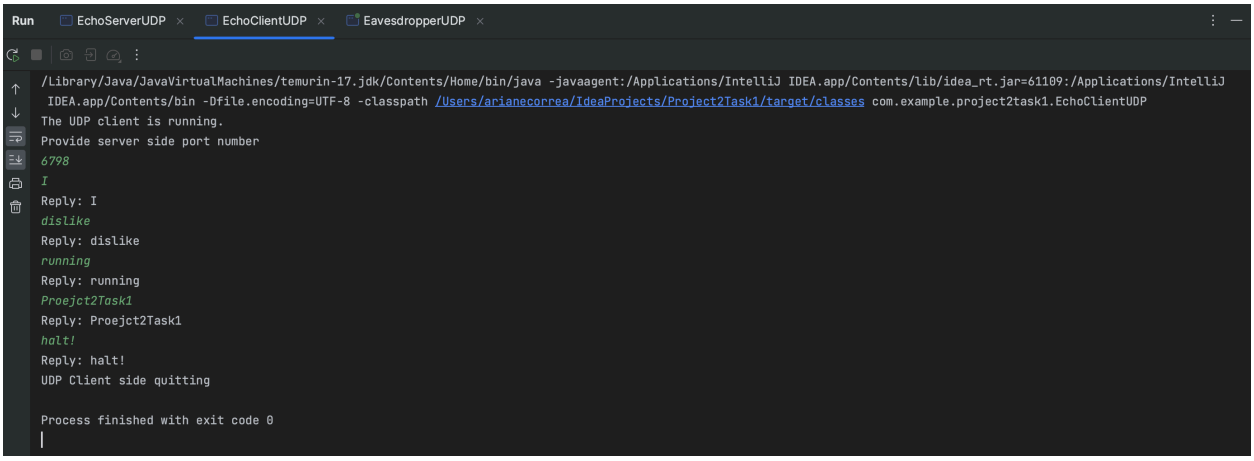
Eavesdropper Console



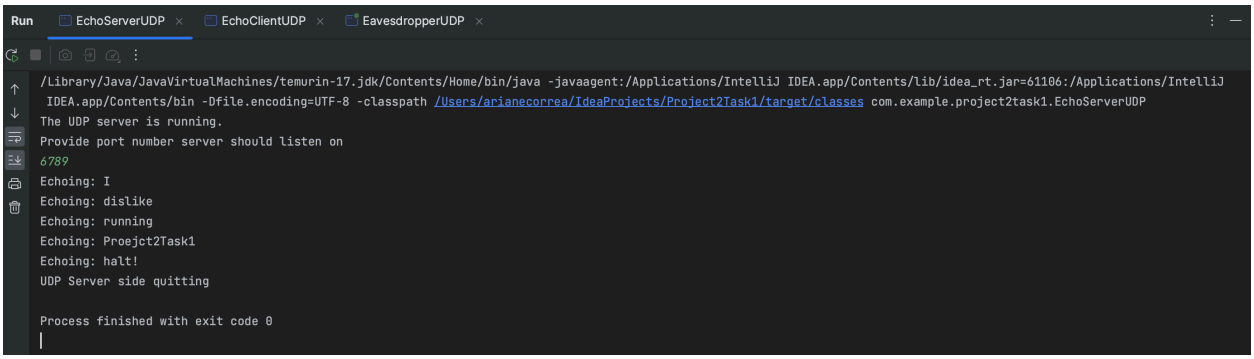
Project2Task1ThreeConsoles

Case 2: *Incorrect Server, word containing 'like', i.e. 'dislike' entered by client -> no change made to the word by the malicious player*

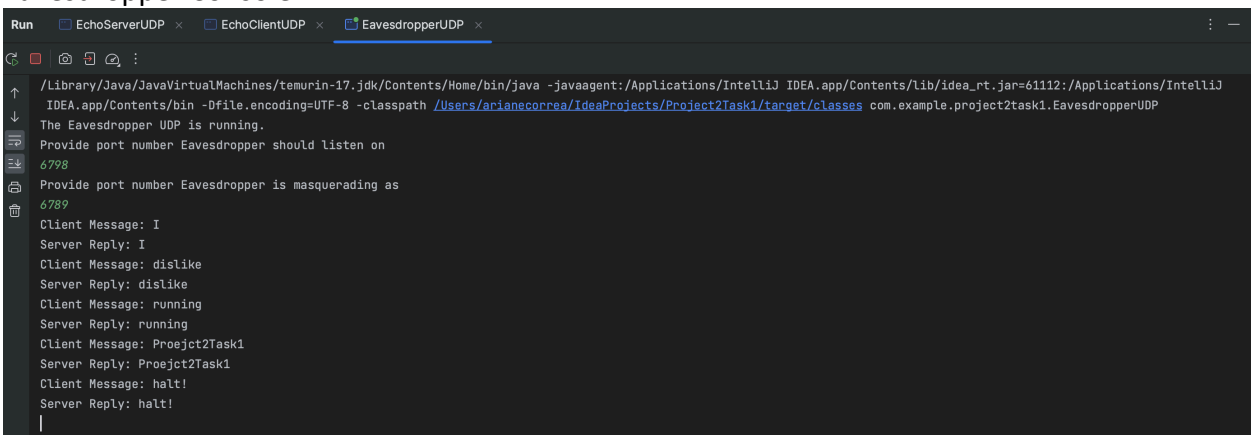
Client Console



Server Console



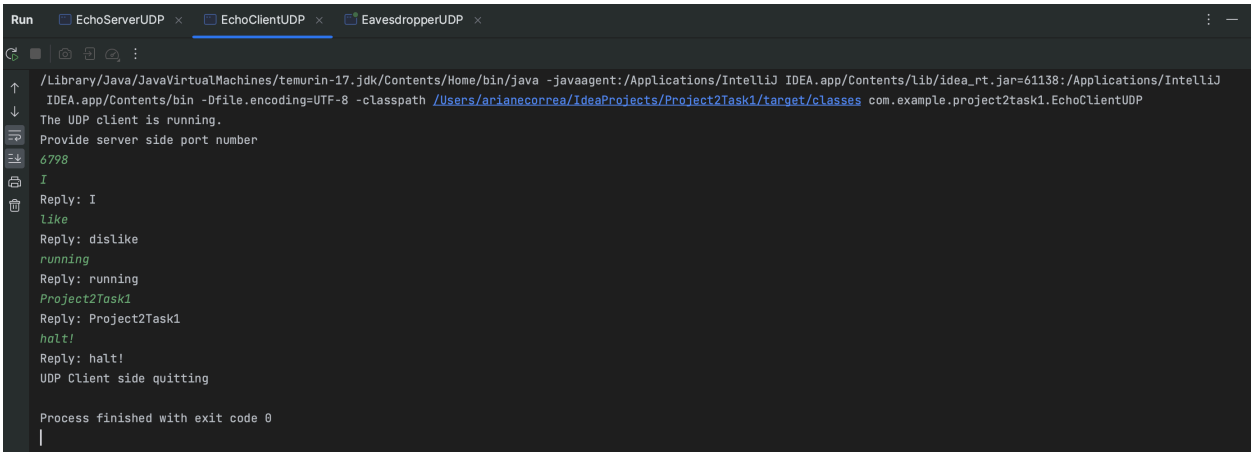
Eavesdropper Console



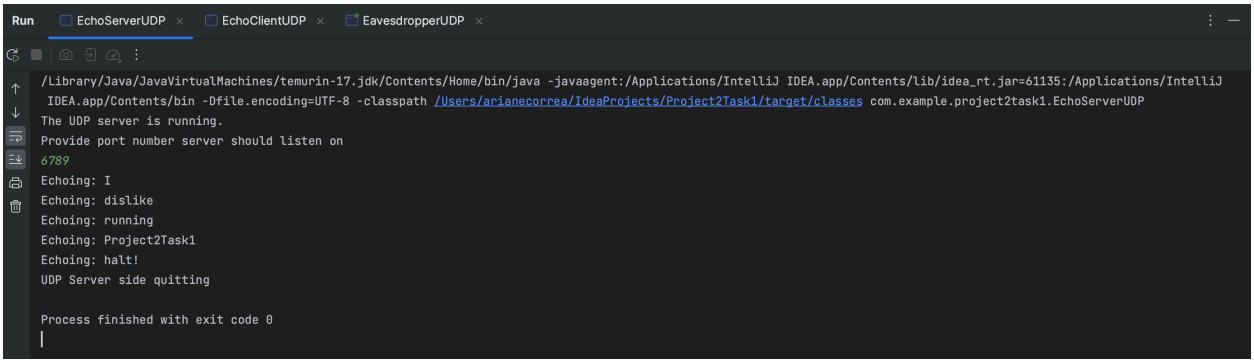
Project2Task1ThreeConsoles

Case 3: *Incorrect Server, the word 'like' is entered by client -> the malicious player changes 'like' to 'dislike'*

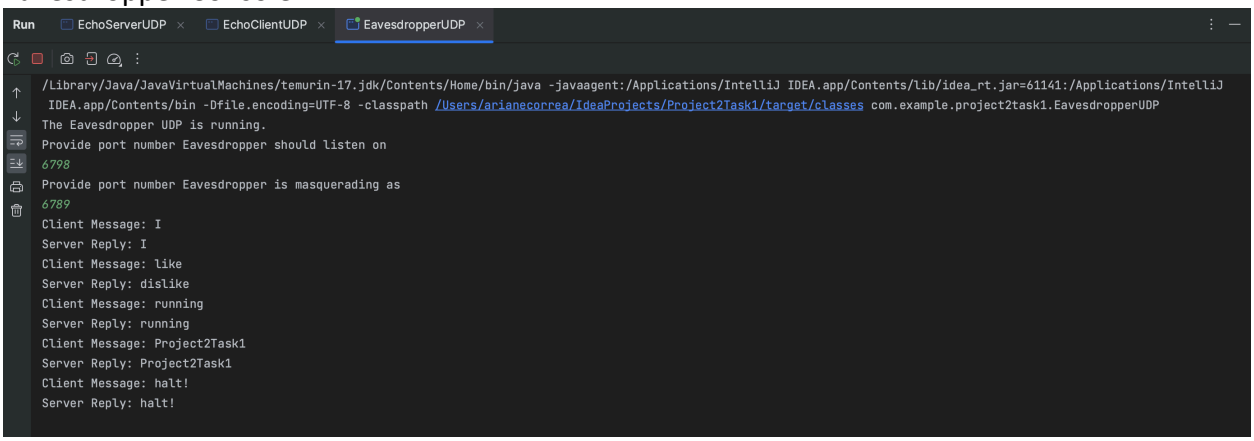
Client Console



Server Console



Eavesdropper Console



95702: Distributed Systems for Information Systems Management

Project 2

Name: Ariane Correa
Andrew Email ID: ajcorrea@andrew.cmu.edu

Task 2:

Project2Task2Client

AddingClientUDP.java | AddingServerUDP.java

```java
// Ariane Correa
// ajcorrea

package com.example.project2task2;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.nio.ByteBuffer;
import java.util.Scanner;

// Reference: EchoClientUDP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)

public class AddingClientUDP {

    2 usages
    static int serverPort;
    5 usages
    static DatagramSocket aSocket = null;

    // args[] gives server hostname and message content
    public static void main(String args[]) {

        try {
            System.out.println("The UDP client is running.");

            Scanner sc = new Scanner(System.in);
            System.out.println("Provide server side port number");
            serverPort = sc.nextInt();  // Read the server's port number from the user.
```

Project2Task2Client (continued)

AddingClientUDP.java | AddingServerUDP.java

```
32
33              aSocket = new DatagramSocket();  // Create a DatagramSocket for sending and receiving UDP packets.
34              String nextLine;
35              BufferedReader typed = new BufferedReader(new InputStreamReader(System.in));
36
37              while ((nextLine = typed.readLine()) != null)
38              {
39
40                  // If "halt!" is received, close the client socket
41                  if (nextLine.equalsIgnoreCase( anotherString: "halt!")) {
42                      System.out.println("UDP Client side quitting");
43                      break;
44                  }
45                  // Else, send value to server
46                  else {
47                      int x = add(Integer.parseInt(nextLine));
48                      System.out.println("The server returned " + x + ".");
49                  }
50              }
51          }
52          catch (SocketException e) {
53              System.out.println("Socket: " + e.getMessage()); // Handle SocketException if it occurs.
54          }
55          catch (IOException e) {
56              System.out.println("IO: " + e.getMessage()); // Handle IOException if it occurs.
57          }
58          finally {
59              if (aSocket != null) aSocket.close(); // Close the DatagramSocket when done.
60          }
61      }
```

Project2Task2Client (continued)

AddingClientUDP.java | AddingServerUDP.java

```
62
        1 usage
63      public static int add(int i) throws IOException {
64
65          // Define the server host (in this case, localhost).
66          InetAddress aHost = InetAddress.getByName( host: "localhost");
67
68          // Convert user input to bytes to send via UDP
69          byte[] m = String.valueOf(i).getBytes();
70
71          // Create a DatagramPacket for sending data to the server.
72          DatagramPacket request = new DatagramPacket(m, m.length, aHost, serverPort);
73          aSocket.send(request); // Send the data to the server.
74
75          // Initialize a new buffer that will store contents of the reply message
76          byte[] buffer = new byte[1000];
77
78          // Create a DatagramPacket for receiving data from the server.
79          DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
80          aSocket.receive(reply); // Receive data from the server.
81
82          // Use reply.getLength() to determine the actual length of the received message
83          byte[] replyArray = new byte[reply.getLength()];
84
85          // Copy relevant reply data into the new array based on reply length
86          System.arraycopy(reply.getData(), srcPos: 0, replyArray, destPos: 0, reply.getLength());
87
88          // Convert the byte array back to an integer using ByteBuffer
89          // Reference Code: https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/nio/ByteBuffer.html
90          int n = ByteBuffer.wrap(replyArray).getInt();
91          return n;
92      }
93  }
```

Project2Task2Server

AddingClientUDP.java | AddingServerUDP.java

```java
// Ariane Correa
// ajcorrea

package com.example.project2task2;

import java.io.IOException;
import java.math.BigInteger;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.nio.ByteBuffer;
import java.util.Scanner;

// Reference: EchoServerUDP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)

public class AddingServerUDP {
    3 usages
    static int i = 0;

    public static void main(String args[]) {

        System.out.println("The UDP server is running.");

        DatagramSocket aSocket = null; // Declare a DatagramSocket for UDP communication.

        // We declare a byte array that will receive client messages
        byte[] buffer = new byte[1000];

        try {
            Scanner sc = new Scanner(System.in);
            System.out.println("Provide port number server should listen on");
            int serverPort = sc.nextInt();  // Read the port number from the user.
            aSocket = new DatagramSocket(serverPort); // Create a DatagramSocket bound to the specified port.
```

Project2Task2Server (continued)

AddingClientUDP.java | AddingServerUDP.java

```
33              // Create a new datagram packet to fetch the request
34              DatagramPacket request = new DatagramPacket(buffer, buffer.length);
35
36              while (true) {
37
38                  aSocket.receive(request); // Receive a UDP packet and store it in 'request'.
39
40                  byte[] replyArray = new byte[request.getLength()];
41                  // Copy the message to the reply array which is the length of the request
42                  System.arraycopy(request.getData(), srcPos: 0, replyArray, destPos: 0, request.getLength());
43                  String requestString = new String(replyArray);
44
45                  byte[] byteArray;
46                  System.out.println("Adding " + requestString + " to " + i);
47
48                  // Add the value received by client
49                  int n = add(Integer.parseInt(requestString));
50                  System.out.println("Returning sum of " + n + " to client");
51
52                  // Convert the value of ByteBuffer to return back to Client
53                  // Reference Code: https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/nio/ByteBuffer.html
54                  byteArray = ByteBuffer.allocate( capacity: 4).putInt(n).array();
55
56                  // Create a new datagram packet with the newly created reply array, request hosts and port array
57                  DatagramPacket reply = new DatagramPacket(byteArray,
58                          byteArray.length, request.getAddress(), request.getPort());
59                  aSocket.send(reply); // Send a reply back to the client.
60              }
61          }
62          catch (SocketException e) {
63              System.out.println("Socket: " + e.getMessage()); // Handle SocketException if it occurs.
64          }
```

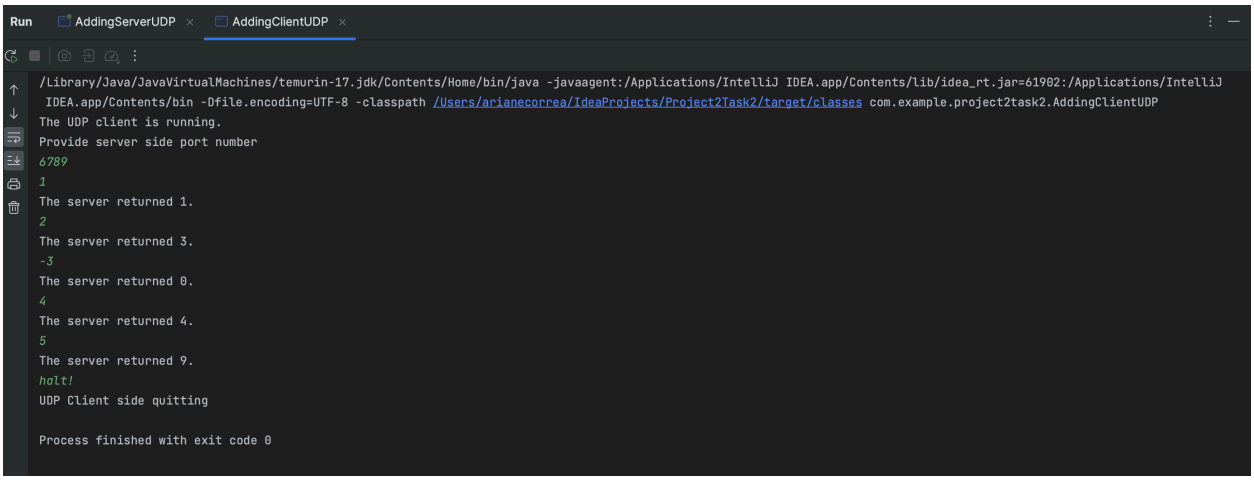
Project2Task2Server (continued)

AddingClientUDP.java | AddingServerUDP.java

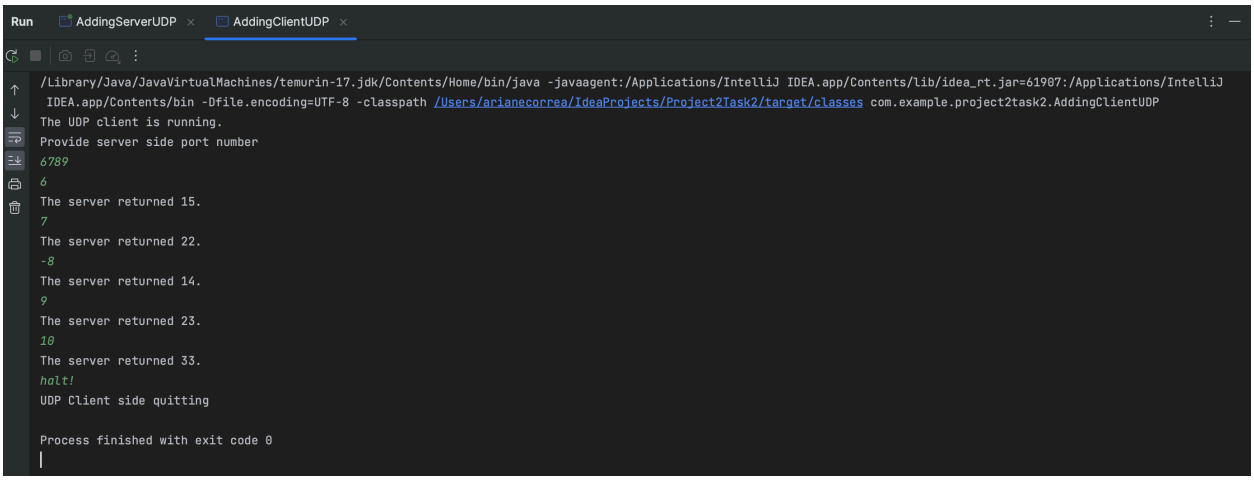
```
65          catch (IOException e) {
66              System.out.println("IO: " + e.getMessage()); // Handle IOException if it occurs.
67          }
68          finally {
69              if (aSocket != null) aSocket.close(); // Close the DatagramSocket when done.
70          }
71      }
72
73      // Adds the value received by the client to global sum variable
        1 usage
74      public static int add(int n) {
75          i += n;
76          return i;
77      }
78
79  }
80
```

Project2Task2ClientConsole

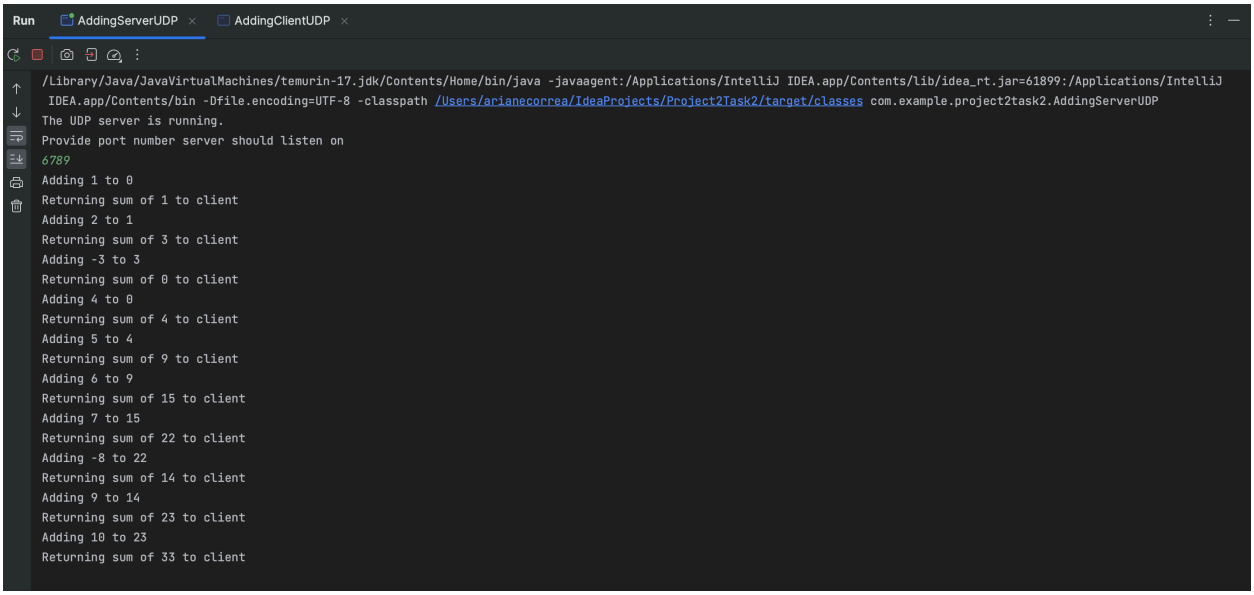
Case 1: *We start the Client and enter 5 integer values (1,2,-3,4,5) and then halt! the Client*



Case 2: *We restart the Client and enter 5 more integer values (6,7,-8,9,10) and then halt! the Client again*



Project2Task2ServerConsole



95702: Distributed Systems for Information Systems Management

Project 2

Name: Ariane Correa
Andrew Email ID: ajcorrea@andrew.cmu.edu

Task 3:

Project2Task3Client

RemoteVariableClientUDP.java | RemoteVariableServerUDP.java

```java
// Ariane Correa
// ajcorrea

package com.example.project2task3;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketException;
import java.nio.ByteBuffer;
import java.util.Scanner;

// Reference: EchoClientUDP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)

public class RemoteVariableClientUDP {

    2 usages
    static int serverPort;
    3 usages
    static DatagramSocket aSocket = null;

    // args[] gives server hostname and message content
    public static void main(String args[]) {

        try {
            System.out.println("The UDP client is running.");

            Scanner sc = new Scanner(System.in);
            System.out.println("Provide server side port number");
            serverPort = sc.nextInt();  // Read the server's port number from the user.

            aSocket = new DatagramSocket(); // Create a DatagramSocket for sending and receiving UDP packets.
```

Project2Task3Client (continued)

RemoteVariableClientUDP.java × RemoteVariableServerUDP.java

```
34
35          while (true) {
36
37              // Display Menu of Operation Options to Client Console
38              System.out.println("1. Add a value to your sum.\n" +
39                      "2. Subtract a value from your sum.\n" +
40                      "3. Get your sum.\n" +
41                      "4. Exit client");
42
43              int option = sc.nextInt();
44              int value = 0;
45              String op = "";
46
47              // Check the user's choice and set the operation and value accordingly
48              if (option == 1) {
49                  System.out.println("Enter value to add:");
50                  value = sc.nextInt();
51                  op = "add";
52              }
53
54              if (option == 2) {
55                  System.out.println("Enter value to subtract:");
56                  value = sc.nextInt();
57                  op = "substract";
58              }
59
60              if (option == 3)
61                  op = "get";
62
63              if (option == 4) {
64                  System.out.println("Client side quitting. The remote variable server is still running.");
65                  break;
66              }
67
68              System.out.println("Enter your ID:");
```

Project2Task3Client (continued)

RemoteVariableClientUDP.java | RemoteVariableServerUDP.java

```
                int userId = sc.nextInt();

                String message = userId + "," + op + "," + value;

                if (message != null) {
                    int x = result(message);
                    System.out.println("The server returned " + x);
                }

            }
        }
        catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage()); // Handle SocketException if it occurs.
        }
        catch (IOException e) {
            System.out.println("IO: " + e.getMessage()); // Handle IOException if it occurs.
        }
    }

    // Method that returns the input from the server
    1 usage
    public static int result(String x) throws IOException {

        // Define the server host (in this case, localhost).
        InetAddress aHost = InetAddress.getByName( host: "localhost");

        // Convert user input to bytes to send via UDP
        byte[] m = String.valueOf(x).getBytes();

        // Create a DatagramPacket for sending data to the server.
        DatagramPacket request = new DatagramPacket(m, m.length, aHost, serverPort);
        aSocket.send(request); // Send the data to the server.

        // Initialize a new buffer that will store contents of the reply message
        byte[] buffer = new byte[1000];

        // Create a DatagramPacket for receiving data from the server.
        DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
        aSocket.receive(reply); // Receive data from the server.

        // Use reply.getLength() to determine the actual length of the received message
        byte[] replyArray = new byte[reply.getLength()];

        // Copy relevant reply data into the new array based on reply length
        System.arraycopy(reply.getData(), srcPos: 0, replyArray, destPos: 0, reply.getLength());

        // Convert the byte array back to an integer using ByteBuffer
        // Reference Code: https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/nio/ByteBuffer.html
        int n = ByteBuffer.wrap(replyArray).getInt();
        return n;

    }
}
```

Project2Task3Server

RemoteVariableClientUDP.java | RemoteVariableServerUDP.java

```java
// Ariane Correa
// ajcorrea

package com.example.project2task3;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.nio.ByteBuffer;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

// Reference: EchoServerUDP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)

public class RemoteVariableServerUDP {

    //static int i = 0;

    // Map to store user IDs and their corresponding values
    11 usages
    static Map<Integer, Integer> UserID_Value_Map = new HashMap<>();

    public static void main(String args[]) {

        System.out.println("The UDP server is running.");

        DatagramSocket aSocket = null; // Declare a DatagramSocket for UDP communication.

        // We declare a byte array that will receive client messages
        byte[] buffer = new byte[1000];

        try {

```

Project2Task3Server (continued)

RemoteVariableClientUDP.java | RemoteVariableServerUDP.java

```
35            Scanner sc = new Scanner(System.in);
36            System.out.println("Provide port number server should listen on");
37            int serverPort = sc.nextInt();  // Read the port number from the user.
38            aSocket = new DatagramSocket(serverPort); // Create a DatagramSocket bound to the specified port.
39            // Create a new datagram packet to fetch the request
40            DatagramPacket request = new DatagramPacket(buffer, buffer.length);
41
42            while (true) {
43
44                aSocket.receive(request); // Receive a UDP packet and store it in 'request'.
45
46                byte[] replyArray = new byte[request.getLength()];
47                // Copy the message to the reply array which is the length of the request
48                System.arraycopy(request.getData(), srcPos: 0, replyArray, destPos: 0, request.getLength());
49                String requestString = new String(replyArray);
50
51                // Use the delimiter to fetch different values
52                String userId = requestString.split( regex: ",")[0];
53                String operation = requestString.split( regex: ",")[1];
54                String value = requestString.split( regex: ",")[2];
55
56                byte[] byteArray;
57                System.out.println("User " + userId + " requested " + operation + " operation.");
58
59                // Initialize integer variable that will store the updated value after operation is performed on value
60                int n = 0;
61
62                // If the operation selected is Addition
63                if (operation.equalsIgnoreCase( anotherString: "add")) {
64                    n = add(Integer.parseInt(userId), Integer.parseInt(value));
65                }
66                // If the operation selected is Subtraction
67                else if (operation.equalsIgnoreCase( anotherString: "substract")) {
68                    n = substract(Integer.parseInt(userId), Integer.parseInt(value));
69                }
```

Project2Task3Server (continued)

RemoteVariableClientUDP.java | RemoteVariableServerUDP.java

```
70                  // If the operation selected is Get
71                  else if (operation.equalsIgnoreCase( anotherString: "get")) {
72                      n = get(Integer.parseInt(userId));
73                  }
74
75                  // Print to Console
76                  System.out.println("Returning " + n + " to client");
77
78                  // Convert the value of ByteBuffer to return back to Client
79                  // Reference Code: https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/nio/ByteBuffer.html
80                  byteArray = ByteBuffer.allocate( capacity: 4).putInt(n).array();
81
82                  // Create a new datagram packet with the newly created reply array, request hosts and port array
83                  DatagramPacket reply = new DatagramPacket(byteArray,
84                          byteArray.length, request.getAddress(), request.getPort());
85                  aSocket.send(reply); // Send a reply back to the client.
86
87                  // If the user enters halt!, Close the Server Socket
88                  if (requestString.equalsIgnoreCase( anotherString: "halt!")) {
89                      System.out.println("UDP Server side quitting");
90                      break;
91                  }
92              }
93          }
94          catch (SocketException e) {
95              System.out.println("Socket: " + e.getMessage()); // Handle SocketException if it occurs.
96          }
97          catch (IOException e) {
98              System.out.println("IO: " + e.getMessage()); // Handle IOException if it occurs.
99          }
100         finally {
101             if (aSocket != null) aSocket.close(); // Close the DatagramSocket when done.
102         }
103     }
104
```

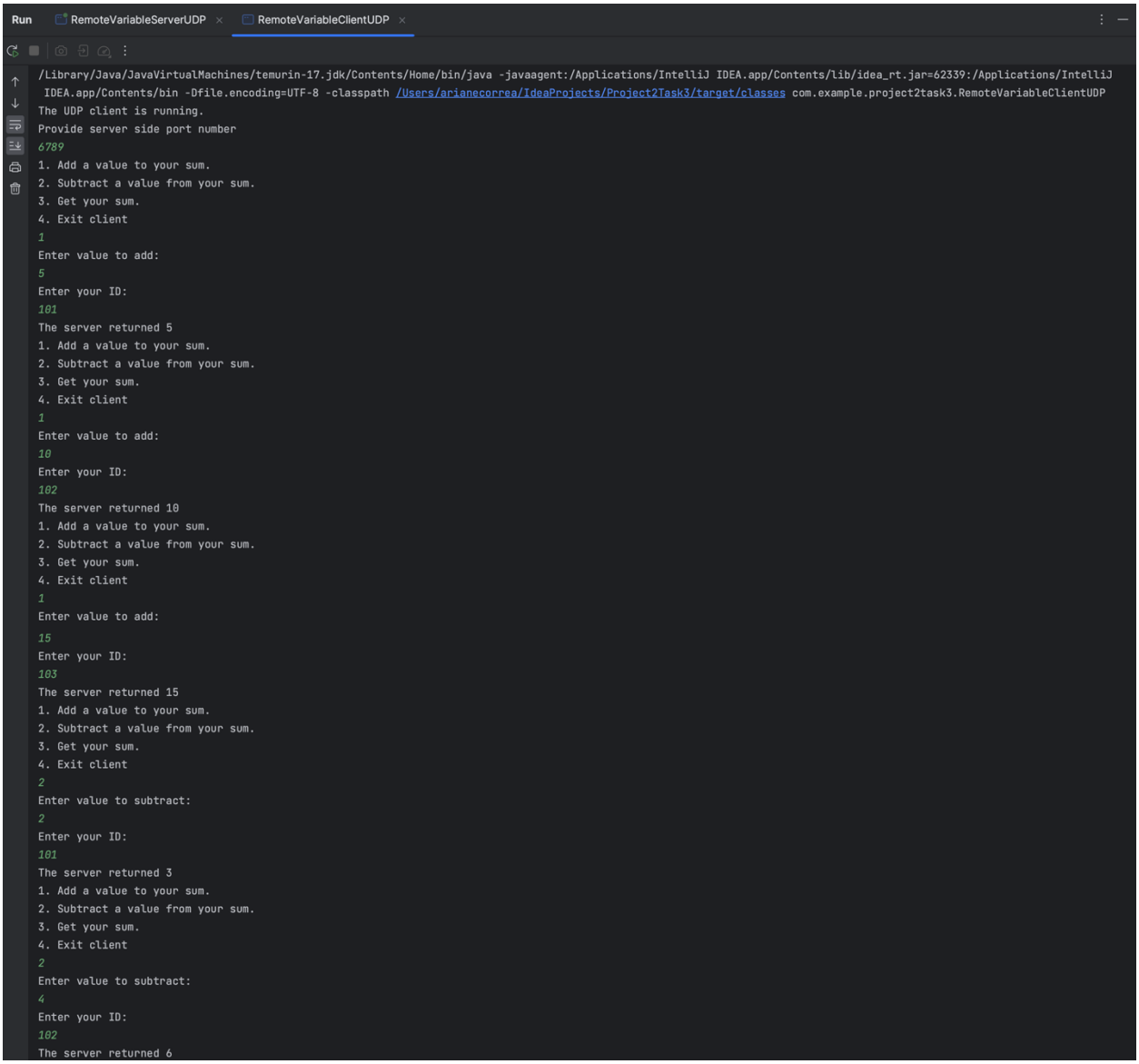
Project2Task3Server (continued)

RemoteVariableClientUDP.java | RemoteVariableServerUDP.java

```
105         // Adds value received to the specific user
            1 usage
106         public static int add(int userId, int n) {
107             int result;
108             if (UserID_Value_Map.containsKey(userId)) {
109                 result = UserID_Value_Map.get(userId) + n;
110                 UserID_Value_Map.put(userId, result);
111             }
112             else {
113                 result = n;
114                 UserID_Value_Map.put(userId, n);
115             }
116             return result;
117         }
118
119         // Subtracts value received from the specific user
            1 usage
120         public static int substract(int userId, int n) {
121             int result;
122             if (UserID_Value_Map.containsKey(userId)) {
123                 result = UserID_Value_Map.get(userId) - n;
124                 UserID_Value_Map.put(userId, result);
125             }
126             else {
127                 result = n;
128                 UserID_Value_Map.put(userId, n);
129             }
130             return result;
131         }
132
133         //  Gets value received from the specific user
            1 usage
134         public static int get(int userId) {
135             int result;
136             if (UserID_Value_Map.containsKey(userId)) {
137                 result = UserID_Value_Map.get(userId);
138             }
139             else {
140                 result = 0;
141                 UserID_Value_Map.put(userId, 0);
142             }
143             return result;
144         }
145     }
146
```

Initially, we start the Client and Server Sockets. Three different clients interact with the server using distinct User IDs. Each performs an add, subtract and a get operation. The client is then halted.

Project2Task3ClientConsole

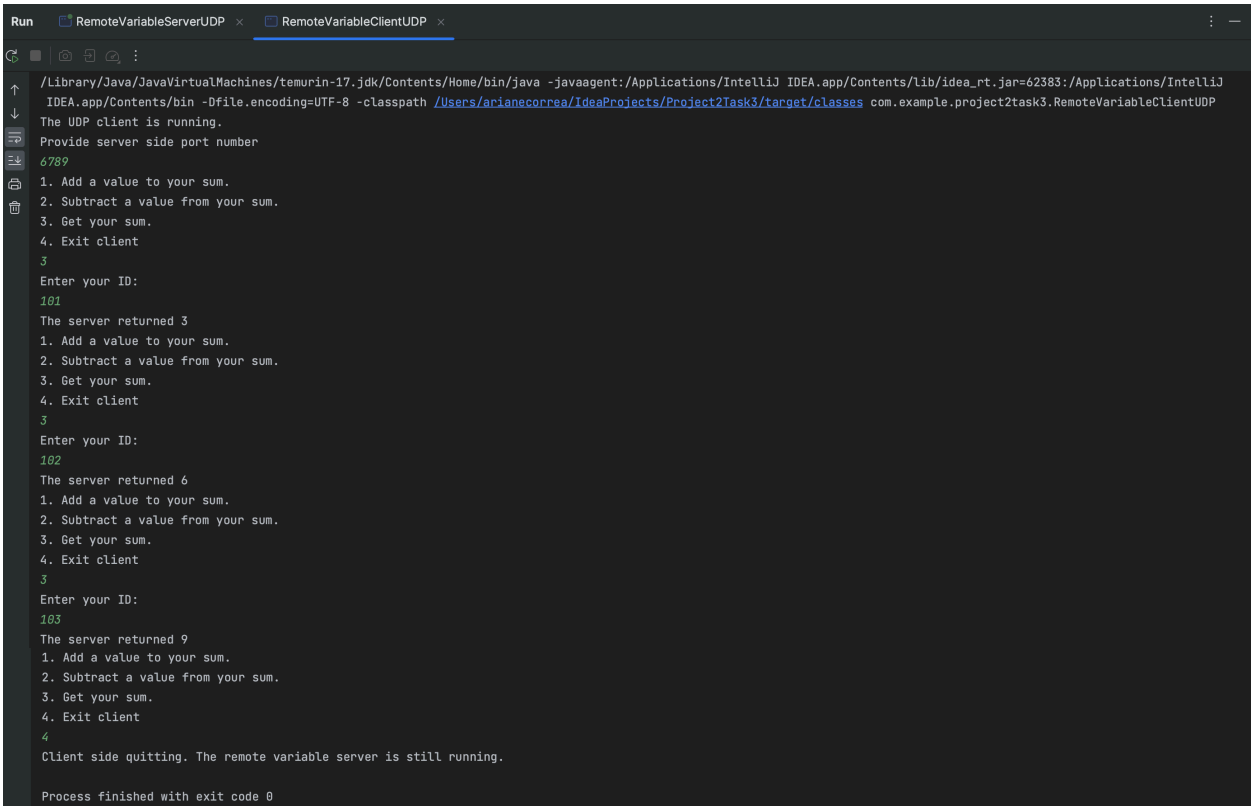


Project2Task3ClientConsole (continued)

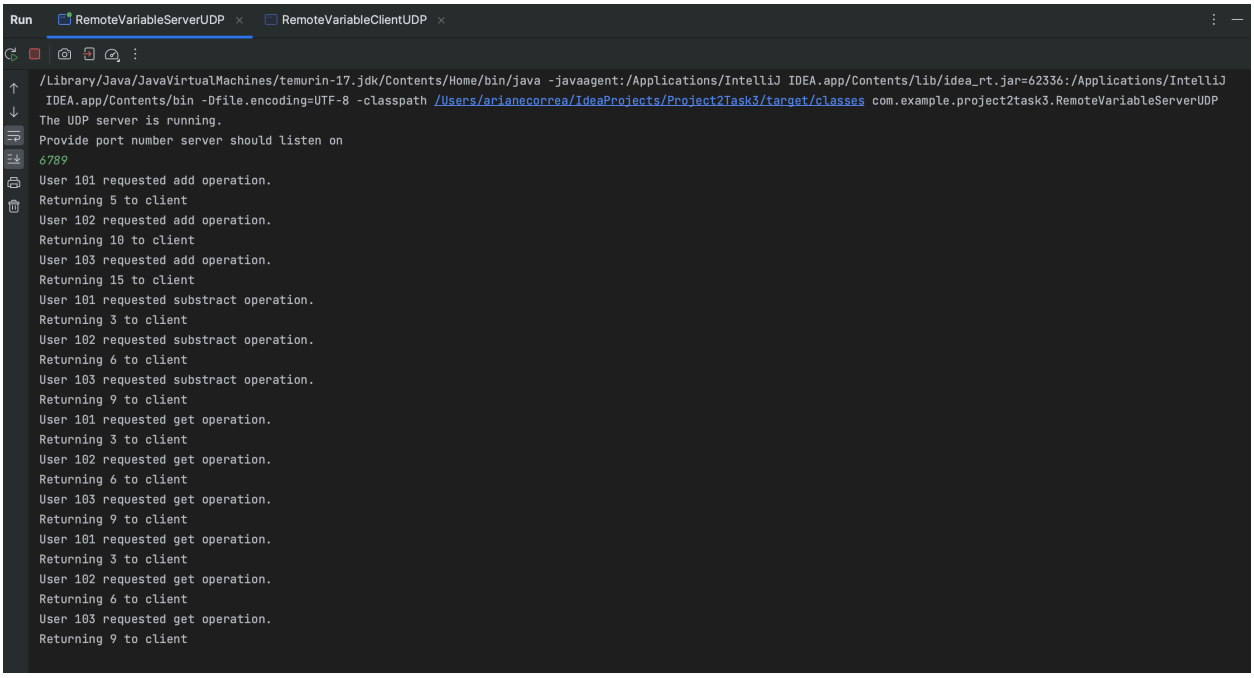
```
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
2
Enter value to subtract:
6
Enter your ID:
103
The server returned 9
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
101
The server returned 3
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
102
The server returned 6
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
103
The server returned 9
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
4
Client side quitting. The remote variable server is still running.

Process finished with exit code 0
```

The Client Socket is now re-started. The three clients interact with the server via get requests. The client is then halted again.

Project2Task3ClientConsole



Project2Task3ServerConsole



95702: Distributed Systems for Information Systems Management

Project 2

Name: Ariane Correa
Andrew Email ID: ajcorrea@andrew.cmu.edu

Task 4:

Project2Task4Client

RemoteVariableServerTCP.java | RemoteVariableClientTCP.java

```java
// Ariane Correa
// ajcorrea

package com.example.project2task4;

import java.io.*;
import java.net.*;
import java.nio.ByteBuffer;
import java.util.Scanner;

// Reference: EchoClientUDP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
// Reference: EchoClientTCP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)

public class RemoteVariableClientTCP {

    2 usages
    static int serverPort;
    6 usages
    static Socket clientSocket = null; // Declare and Initialize Client Socket object

    // args[] gives server hostname and message content
    public static void main(String args[]) {

        System.out.println("The TCP client is running.");
        Scanner sc = new Scanner(System.in);
        System.out.println("Provide server side port number");

        try {
            serverPort = sc.nextInt();  // Read the server's port number from the user.
            clientSocket = new Socket( host: "localhost", serverPort);

```

```
            while (!clientSocket.isClosed()) {

                // Display Menu of Operation Options to Client Console
                System.out.println("1. Add a value to your sum.\n" +
                        "2. Subtract a value from your sum.\n" +
                        "3. Get your sum.\n" +
                        "4. Exit client");

                int option = sc.nextInt();
                int value = 0;
                String op = "";

                // Check the user's choice and set the operation and value accordingly
                if (option == 1) {
                    System.out.println("Enter value to add:");
                    value = sc.nextInt();
                    op = "add";
                }

                if (option == 2) {
                    System.out.println("Enter value to subtract:");
                    value = sc.nextInt();
                    op = "substract";
                }

                if (option == 3)
                    op = "get";

                if (option == 4) {
                    System.out.println("Client side quitting. The remote variable server is still running.");
                    break;
                }

                System.out.println("Enter your ID:");
                int userId = sc.nextInt();

                String message = userId + "," + op + "," + value;

                if (message != null) {
                    int x = result(message);
                    System.out.println("The server returned " + x);
                }
            }
        }
        catch (IOException e) {
            // Handle IO Exception by displaying an error message
            System.out.println("IO Exception: " + e.getMessage());
        }
        finally {
            try {
                // Ensure that the clientSocket is closed, if it's not null
                if (clientSocket != null) {
                    clientSocket.close();
                }
            }
            catch (IOException e) {
                // Handle any potential IOException that may occur when closing the socket
                // This is typically ignored, as the main goal is to ensure the socket is closed
            }
        }
    }

```

```
92      // Method that returns the input from the server
        1 usage
93      public static int result(String str) throws IOException {
94
95          // Create a BufferedReader to read the response from the server
96          BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
97
98          // Create a PrintWriter to send the message to the server
99          PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));
100
101         // Send the message to the server
102         out.println(str);
103         out.flush();
104
105         // Read the response from the server
106         String line = in.readLine();
107
108         // Print the received response to the client's console
109         System.out.println("Received: " + line);
110
111         // Parse the response as an integer and return it
112         return Integer.parseInt(line);
113     }
114
115 }
116
```

Project2Task4Server

RemoteVariableServerTCP.java | RemoteVariableClientTCP.java

```java
// Ariane Correa
// ajcorrea

package com.example.project2task4;

import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.*;
import java.nio.ByteBuffer;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.util.TreeMap;

// Reference: EchoServerUDP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
// Reference: EchoServerTCP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)

public class RemoteVariableServerTCP {

    // Map to store user IDs and their corresponding values
    11 usages
    static Map<Integer, Integer> UserID_Value_Map = new TreeMap<>();

    public static void main(String args[]) {

        Socket clientSocket = null; // Declare and Initialize Client Socket
        System.out.println("The TCP server is running.");

        try {

            Scanner sc = new Scanner(System.in);
            System.out.println("Provide port number server should listen on");
            int serverPort = sc.nextInt();  // Read the port number from the user.
            ServerSocket listenSocket = new ServerSocket(serverPort); //Initialize Server Socket
```

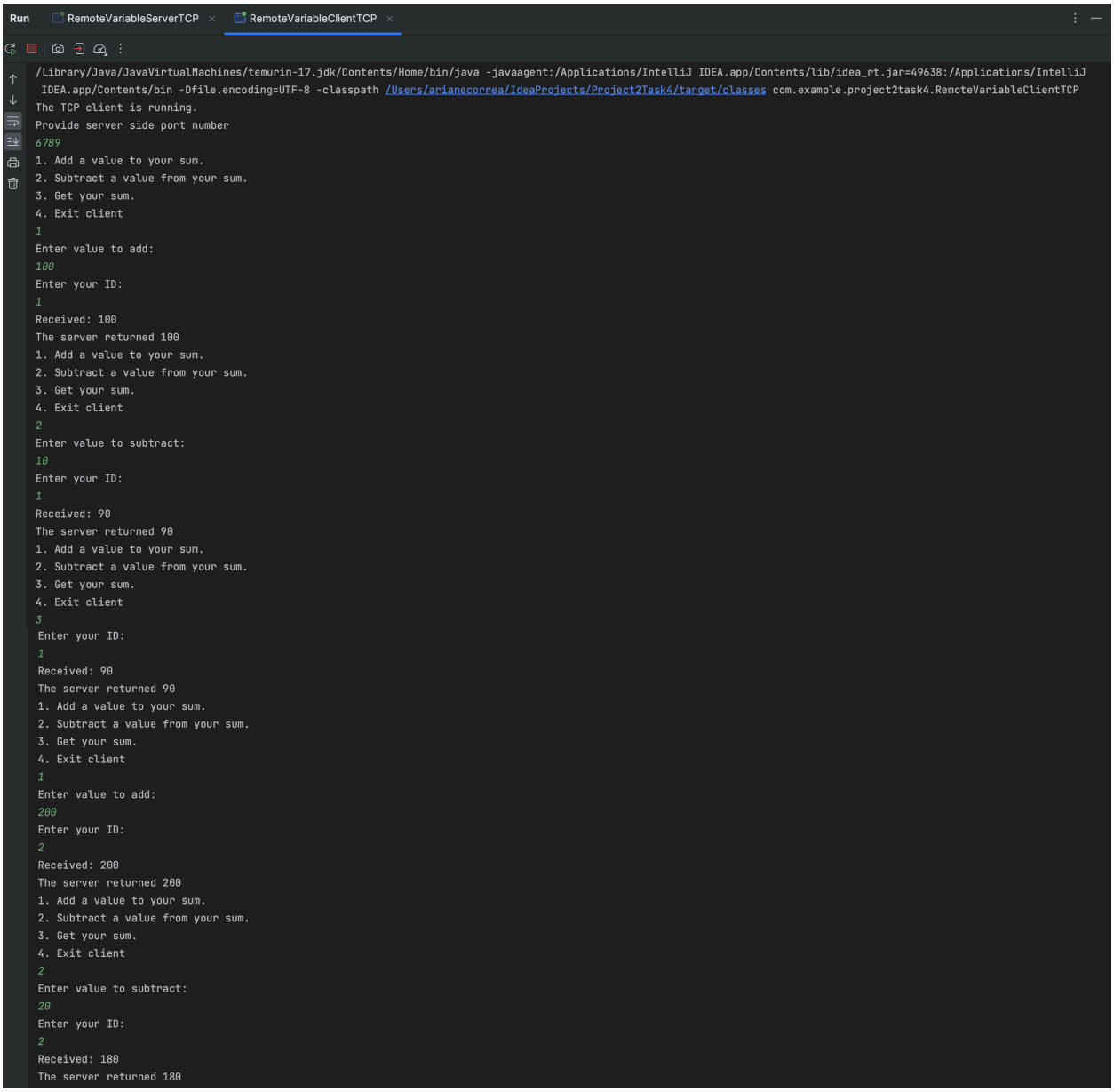
```java
        /*
         * Block waiting for a new connection request from a client.
         * When the request is received, "accept" it, and the rest
         * the tcp protocol handshake will then take place, making
         * the socket ready for reading and writing.
         */
        clientSocket = listenSocket.accept();
        // If we get here, then we are now connected to a client.

        // Set up "in" to read from the client socket
        Scanner in;
        in = new Scanner(clientSocket.getInputStream());

        // Set up "out" to write to the client socket
        PrintWriter out;
        out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));

        /*
         * Forever,
         *   read a line from the socket
         *   print it to the console
         *   echo it (i.e. write it) back to the client
         */

        while (!clientSocket.isClosed())  {

            // Check if there is a line available to be read
            // If not, it is because Client side has quit, but server can continue running
            if (in.hasNextLine()) {
                String requestString = in.nextLine();

                // Use the delimiter to fetch different values
                String userId = requestString.split( regex: ",")[0];
                String operation = requestString.split( regex: ",")[1];
                String value = requestString.split( regex: ",")[2];

                System.out.println("User " + userId + " requested " + operation + " operation.");

                // Initialize integer variable that will store the updated value
                // after the operation is performed on value
                int n = 0;

                // If the operation selected is Addition
                if (operation.equalsIgnoreCase( anotherString: "add")) {
                    n = add(Integer.parseInt(userId), Integer.parseInt(value));
                }
                // If the operation selected is Subtraction
                else if (operation.equalsIgnoreCase( anotherString: "substract")) {
                    n = substract(Integer.parseInt(userId), Integer.parseInt(value));
                }
                // If the operation selected is Get
                else if (operation.equalsIgnoreCase( anotherString: "get")) {
                    n = get(Integer.parseInt(userId));
                }
```

```
                    // Print to Console
                    System.out.println("Returning " + n + " to client");
                    out.println(n);
                    out.flush();
                }
            }
        }
        catch (IOException e) {
            // Handle IO Exception by displaying an error message
            System.out.println("IO Exception: " + e.getMessage());
        }
        finally {
            try {
                // Ensure that the clientSocket is closed, if it's not null
                if (clientSocket != null) {
                    clientSocket.close();
                }
            }
            catch (IOException e) {
                // Handle any potential IOException that may occur when closing the socket
                // This is typically ignored, as the main goal is to ensure the socket is closed
            }
        }
    }

    // Adds value received to the specific user
    1 usage
    public static int add(int userId, int n) {
        int result;
        if (UserID_Value_Map.containsKey(userId)) {
            result = UserID_Value_Map.get(userId) + n;
            UserID_Value_Map.put(userId, result);
        }
        else {
            result = n;
            UserID_Value_Map.put(userId, n);
        }
        return result;
    }

```

```
    // Subtracts value received from the specific user
    1 usage
    public static int substract(int userId, int n) {
        int result;
        if (UserID_Value_Map.containsKey(userId)) {
            result = UserID_Value_Map.get(userId) - n;
            UserID_Value_Map.put(userId, result);
        }
        else {
            result = n;
            UserID_Value_Map.put(userId, n);
        }
        return result;
    }

    //  Gets value received from the specific user
    1 usage
    public static int get(int userId) {
        int result;
        if (UserID_Value_Map.containsKey(userId)) {
            result = UserID_Value_Map.get(userId);
        }
        else {
            result = 0;
            UserID_Value_Map.put(userId, 0);
        }
        return result;
    }
}
```

Initially, we start the Client and Server Sockets. Three different clients interact with the server using distinct User IDs. Each performs an add, subtract and a get operation. The client is then halted.

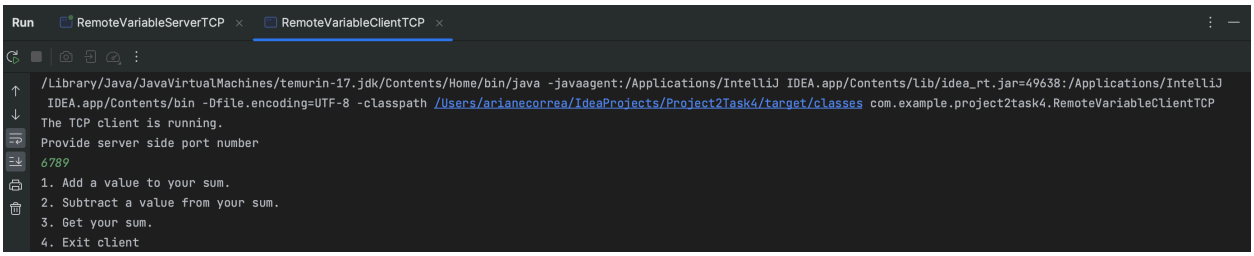
Project2Task4ClientConsole



```
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
2
Received: 180
The server returned 180
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
1
Enter value to add:
300
Enter your ID:
3
Received: 300
The server returned 300
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
2
Enter value to subtract:
30
Enter your ID:
3
Received: 270
The server returned 270
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
3
Received: 270
The server returned 270
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
4
Client side quitting. The remote variable server is still running.

Process finished with exit code 0
```

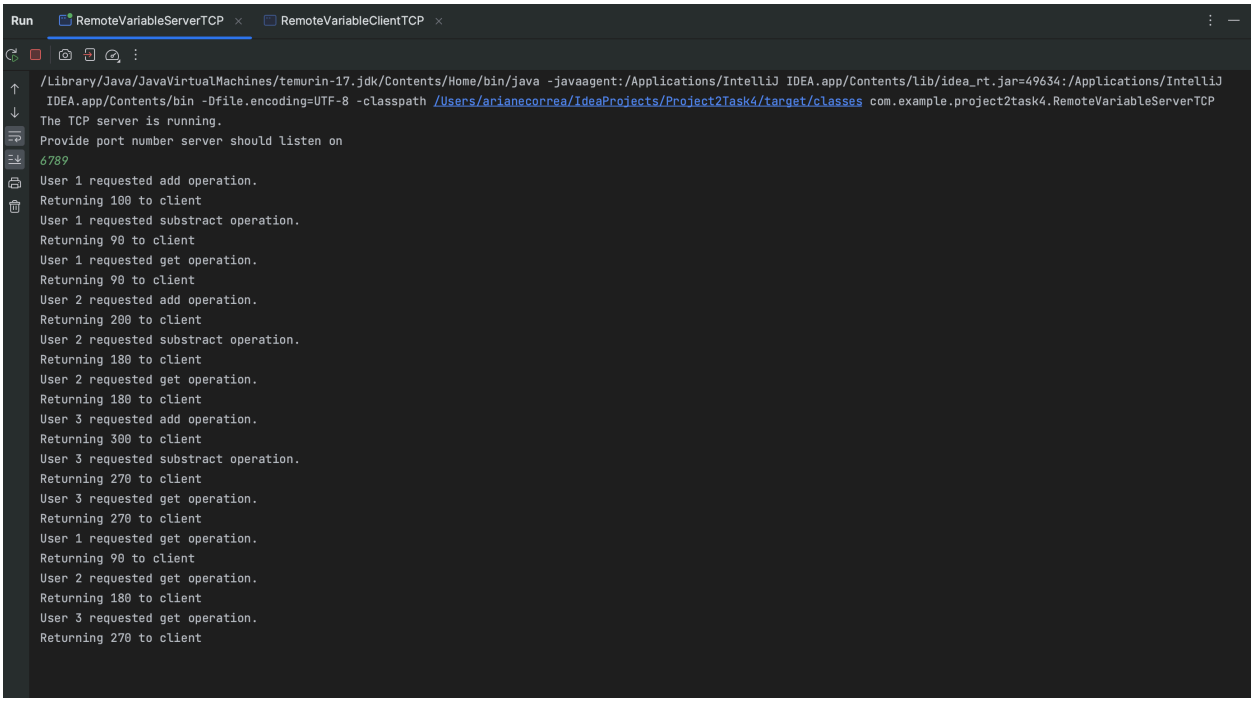
The Client Socket is now re-started. The three clients interact with the server via get requests. The client is then halted again.

Project2Task4ClientConsole



```
3
Enter your ID:
1
Received: 90
The server returned 90
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
2
Received: 180
The server returned 180
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
3
Received: 270
The server returned 270
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
4
Client side quitting. The remote variable server is still running.

Process finished with exit code 0
```

Project2Task4ServerConsole



95702: Distributed Systems for Information Systems Management

Project 2

Name: Ariane Correa
Andrew Email ID: ajcorrea@andrew.cmu.edu

Task 5:

Project2Task5Client

VerifyingServerTCP.java | SigningClientTCP.java

```java
// Ariane Correa
// ajcorrea

package com.example.project2task5;

import java.io.*;
import java.math.BigInteger;
import java.net.InetAddress;
import java.net.Socket;
import java.net.SocketException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Random;
import java.util.Scanner;

// Reference: EchoServerUDP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
// Reference: EchoServerTCP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)

public class SigningClientTCP {

    2 usages
    static int serverPort;
    6 usages
    static Socket clientSocket = null; // Declare and Initialize Client Socket object
    6 usages
    static BigInteger n; // n is the modulus for both the private and public keys
    5 usages
    static BigInteger e; // e is the exponent of the public key
    3 usages
    static BigInteger d; // d is the exponent of the private key

    // args[] gives server hostname
    public static void main(String args[]) {

        System.out.println("The TCP client is running.");
        Scanner sc = new Scanner(System.in);
```

```java
        System.out.println("Provide server side port number");

        try {
            serverPort = sc.nextInt();  // Read the server's port number from the user.
            clientSocket = new Socket(InetAddress.getByName( host: "localhost"), serverPort);

            // Generate keys once
            generateKeys();
            // Generate the userId once
            String id = generateId( key: String.valueOf(e) + String.valueOf(n));

            while (!clientSocket.isClosed()) {

                // Display Menu of Operation Options to Client Console
                System.out.println("1. Add a value to your sum.\n" +
                        "2. Subtract a value from your sum.\n" +
                        "3. Get your sum.\n" +
                        "4. Exit client");

                int option = sc.nextInt();
                int value = 0;
                String op = "";

                // Check the user's choice and set the operation and value accordingly
                if (option == 1) {
                    System.out.println("Enter value to add:");
                    value = sc.nextInt();
                    op = "add";
                }

                if (option == 2) {
                    System.out.println("Enter value to subtract:");
                    value = sc.nextInt();
                    op = "subtract";
                }

                if (option == 3)
                    op = "get";

                if (option == 4) {
                    System.out.println("Client side quitting. The remote variable server is still running.");
                    break;
                }

                //generateKeys();
                //String id = generateId(String.valueOf(e) + String.valueOf(n));

                String message = id + "," + String.valueOf(e) + String.valueOf(n) + "," + op + "," + value;

                if (message != null) {
                    String x = result(message);
                    System.out.println("The server returned " + x);
                }
            }
```

```
        } catch (SocketException e) {
            // Handle Socket Exception by displaying an error message
            System.out.println("Socket: " + e.getMessage());
        } catch (IOException e) {
            // Handle IO Exception by displaying an error message
            System.out.println("IO Exception: " + e.getMessage());
        } catch (NoSuchAlgorithmException ex) {
            // Handle Runtime Exception by displaying an error message
            throw new RuntimeException(ex);
        } finally {
            try {
                // Ensure that the clientSocket is closed, if it's not null
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // Handle any potential IOException that may occur when closing the socket
                // This is typically ignored, as the main goal is to ensure the socket is closed
            }
        }
    }

    // Method that returns the input from the server @throws IOException @throws NoSuchAlgorithmException
    1 usage
    public static String result(String n) throws IOException, NoSuchAlgorithmException {

        // Combine the input 'n' with its cryptographic signature
        String encrpytedMsg = n + "," + sign(n);

        // Set up input and output streams for communication
        BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
        PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));

        // Send the encrypted message to the server
        out.println(encrpytedMsg);
        out.flush();

        // Read the server's response
        String data = in.readLine(); // Reads a line of data from the stream
        System.out.println("Received: " + data);

        // Return the received data
        return data;
    }

```

```java
    // Method to generate public and private keys for user
    1 usage
    public static void generateKeys() {
        // Each public and private key consists of an exponent and a modulus
        // Reference Code: RSAExample.java (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
        Random rnd = new Random();

        // Step 1: Generate two large random prime numbers, p and q.
        // We use 400 bits here, but best practice for security is 2048 bits.
        // Change 400 to 2048, recompile, and run the program again, and you will
        // notice it takes much longer to do the math with that many bits.
        BigInteger p = new BigInteger( bitLength: 400,  certainty: 100, rnd);
        BigInteger q = new BigInteger( bitLength: 400,  certainty: 100, rnd);

        // Step 2: Compute n by the equation n = p * q.
        n = p.multiply(q);

        // Step 3: Compute phi(n) = (p-1) * (q-1)
        BigInteger phi = (p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));

        // Step 4: Select a small odd integer e that is relatively prime to phi(n).
        // By convention, the prime 65537 is used as the public exponent.
        e = new BigInteger( val: "65537");

        // Step 5: Compute d as the multiplicative inverse of e modulo phi(n).
        d = e.modInverse(phi);

        // Step 6: Output the public key (e, n), which is used for encryption.
        System.out.println("Public key = " + e + n);

        // Step 7: Output the private key (d, n), which is used for decryption.
        System.out.println("Private key = " + d + n);
    }

    // Method that returns the input from the server @throws NoSuchAlgorithmException
    1 usage
    public static String generateId(String key) throws NoSuchAlgorithmException {
        // Step 1: Create a MessageDigest instance using SHA-256 hashing algorithm.
        MessageDigest digest = MessageDigest.getInstance( algorithm: "SHA-256");

        // Step 2: Generate a hash of the input 'key'.
        byte[] hashBytes = digest.digest(key.getBytes());

        // Step 3: Create a new byte array to store the final 20 bytes of the hash.
        byte[] finalBytes = new byte[20];

        // Step 4: Copy the least significant 20 bytes of the hash into the final array.
        // This is done to ensure that the output is a 20-byte (160-bit) hash.
        System.arraycopy(hashBytes,  srcPos: hashBytes.length - 20, finalBytes,  destPos: 0,  length: 20);

        // Step 5: Convert the byte array to a hexadecimal string representation.
        return bytesToHex(finalBytes);
    }
```

```java

    // Helper function to convert a byte array to a hexadecimal string
    // Reference Code: https://mkyong.com/java/java-how-to-convert-bytes-to-hex/
    1 usage
    private static String bytesToHex(byte[] bytes) {
        StringBuilder hexString = new StringBuilder();
        for (byte b : bytes) {
            // Convert each byte to a 2-character hexadecimal representation
            String hex = Integer.toHexString( i: 0xFF & b);
            if (hex.length() == 1) {
                // Pad with leading zero if necessary
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }

    // Method to sign the message sent to client using private key @throws UnsupportedEncodingException
    // @throws NoSuchAlgorithmException
    // Reference Code : ShortMessageSign.java (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
    1 usage
    public static String sign(String message) throws UnsupportedEncodingException, NoSuchAlgorithmException {
        // Step 1: Convert the input 'message' to bytes using UTF-8 encoding.
        byte[] bytesOfMessage = message.getBytes( charsetName: "UTF-8");

        // Step 2: Compute the SHA-256 digest of the message.
        MessageDigest md = MessageDigest.getInstance( algorithm: "SHA-256");
        byte[] bigDigest = md.digest(bytesOfMessage);

        // Step 3: Set the most significant byte of the digest to 0 (zero).
        bigDigest[0] = 0;

        // Step 4: Convert the modified digest to a BigInteger.
        BigInteger m = new BigInteger(bigDigest);

        // Step 5: Encrypt the digest using the private key (d, n) with modular exponentiation.
        BigInteger c = m.modPow(d, n);

        // Step 6: Return the encrypted digest as a string in BigInteger format.
        return c.toString();
    }

}
```

Project2Task5Server

VerifyingServerTCP.java | SigningClientTCP.java

```java
// Ariane Correa
// ajcorrea

package com.example.project2task5;

import java.io.*;
import java.math.BigInteger;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.util.TreeMap;

// Reference: EchoServerUDP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
// Reference: EchoServerTCP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)

public class VerifyingServerTCP {
    // Map to store user IDs and their corresponding values
    11 usages
    static Map<String, Integer> UserID_Value_Map = new TreeMap<>();
    5 usages
    static Socket clientSocket = null; // Declare and Initialize Client Socket
    4 usages
    static BigInteger e; // e is the exponent of the public key
    4 usages
    static BigInteger n; // n is the modulus for both the private and public keys
    6 usages
    static String operation;
    4 usages
    static String value;
    5 usages
    static String userId;
```

```java

    public static void main(String args[]) {

        System.out.println("The TCP server is running.");

        try {

            Scanner sc = new Scanner(System.in);
            System.out.println("Provide port number server should listen on");
            int serverPort = sc.nextInt();  // Read the port number from the user.
            ServerSocket listenSocket = new ServerSocket(serverPort); //Initialize Server Socket

            /*
             * Block waiting for a new connection request from a client.
             * When the request is received, "accept" it, and the rest
             * the tcp protocol handshake will then take place, making
             * the socket ready for reading and writing.
             */
            clientSocket = listenSocket.accept();
            // If we get here, then we are now connected to a client.

            // Set up "in" to read from the client socket
            Scanner in;
            in = new Scanner(clientSocket.getInputStream());

            // Set up "out" to write to the client socket
            PrintWriter out;
            out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));

            /*
             * Forever,
             *   read a line from the socket
             *   print it to the console
             *   echo it (i.e. write it) back to the client
             */

            while (true) {
                if (in.hasNextLine()) {
                    String requestString = in.nextLine();

                    if (verify(requestString)) {

                        System.out.println("Signature is verified");

                        System.out.println("User " + userId + " requested " + operation + " operation.");

                        // Initialize integer variable that will store the updated value
                        // after the operation is performed on value
                        int n = 0;

                        // If the operation selected is Addition
                        if (operation.equalsIgnoreCase( anotherString: "add")) {
                            n = add(userId, Integer.parseInt(value));
                        }
                        // If the operation selected is Subtraction
                        else if (operation.equalsIgnoreCase( anotherString: "subtract")) {
                            n = subtract(userId, Integer.parseInt(value));
                        }
                        // If the operation selected is Get
                        else if (operation.equalsIgnoreCase( anotherString: "get")) {
                            n = get(userId);
                        }
```

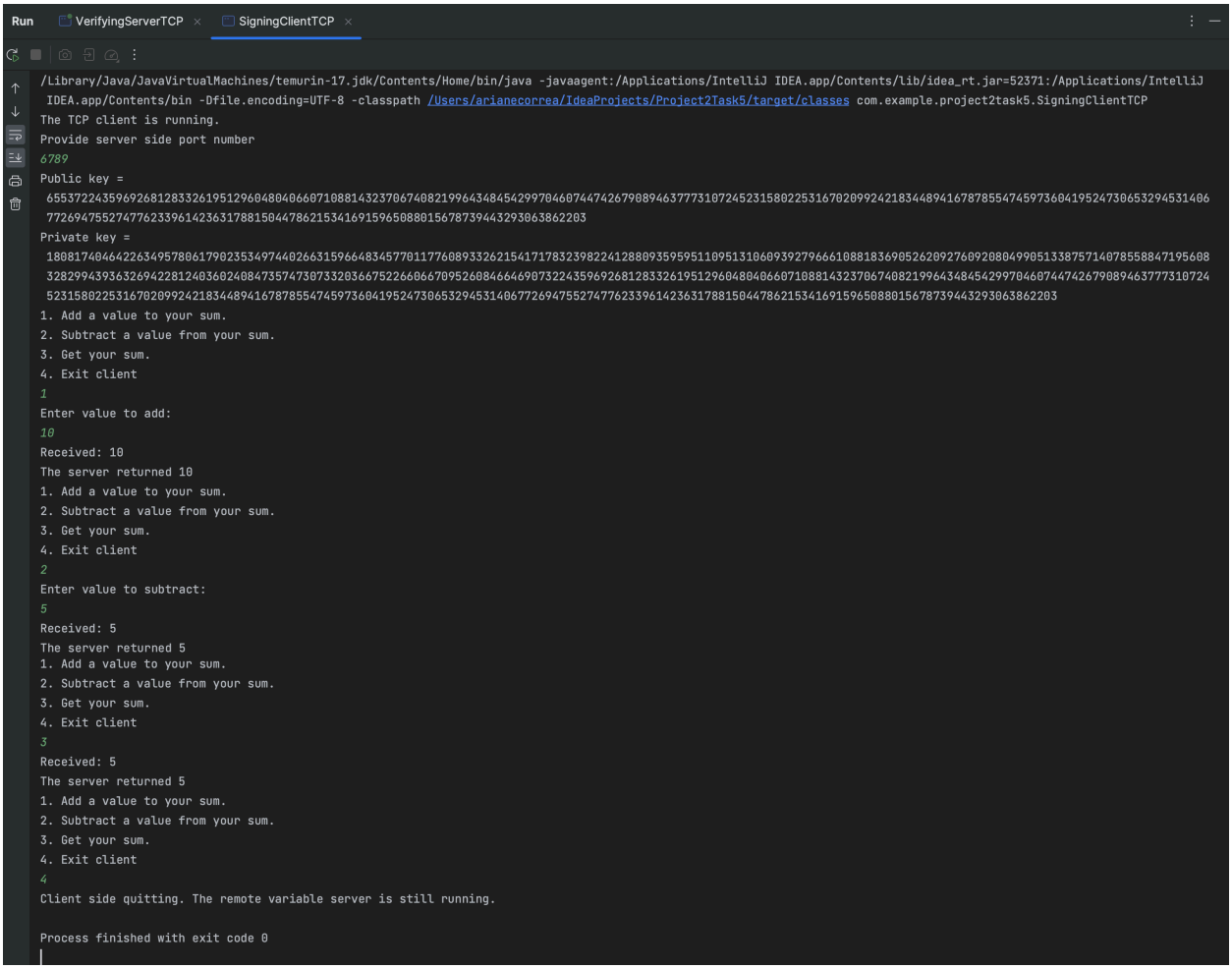
```
                    // Print to Console
                    System.out.println("Returning " + n + " to client");
                    out.println(n);
                    out.flush();
                } else {
                    out.println("Error in request");
                    out.flush();
                }
            }
        }
    }
    catch (SocketException e) {
        // Handle Socket Exception by displaying an error message
        System.out.println("Socket: " + e.getMessage());
    } catch (IOException e) {
        // Handle IO Exception by displaying an error message
        System.out.println("IO Exception: " + e.getMessage());
    } catch (NoSuchAlgorithmException ex) {
        // Handle Runtime Exception by displaying an error message
        throw new RuntimeException(ex);
    } finally {
        try {
            // Ensure that the clientSocket is closed, if it's not null
            if (clientSocket != null) {
                clientSocket.close();
            }
        } catch (IOException e) {
            // Handle any potential IOException that may occur when closing the socket
            // This is typically ignored, as the main goal is to ensure the socket is closed
        }
    }
}

// Adds value received to the specific user
1 usage
public static int add(String userId, int n) {
    int result;
    if (UserID_Value_Map.containsKey(userId)) {
        result = UserID_Value_Map.get(userId) + n;
        UserID_Value_Map.put(userId, result);
    }
    else {
        result = n;
        UserID_Value_Map.put(userId, n);
    }
    return result;
}
```

```java
    // Subtracts value received from the specific user
    1 usage
    public static int subtract(String userId, int n) {
        int result;
        if (UserID_Value_Map.containsKey(userId)) {
            result = UserID_Value_Map.get(userId) - n;
            UserID_Value_Map.put(userId, result);
        }
        else {
            result = n;
            UserID_Value_Map.put(userId, n);
        }
        return result;
    }

    //  Gets value received from the specific user
    1 usage
    public static int get(String userId) {
        int result;
        if (UserID_Value_Map.containsKey(userId)) {
            result = UserID_Value_Map.get(userId);
        }
        else {
            result = 0;
            UserID_Value_Map.put(userId, 0);
        }
        return result;
    }

    // Method to verify if encryption was done correctly @throws NoSuchAlgorithmException
    // @throws UnsupportedEncodingException
    1 usage
    public static boolean verify(String message) throws NoSuchAlgorithmException, UnsupportedEncodingException {

        boolean verify = true;

        // Fetch different values from the string
        String id = message.split( regex: ",")[0];
        String publicKey = message.split( regex: ",")[1];
        operation = message.split( regex: ",")[2];
        value = message.split( regex: ",")[3];
        String sign = message.split( regex: ",")[4];


        // Fetch the public and modulus , public key is first 6 characters
        e = new BigInteger(publicKey.substring(0, 5));
        n = new BigInteger(publicKey.substring( beginIndex: 5));

        System.out.println("public key recieved from client " + e+n);

        // Validate id
        String validId = validateId(publicKey);

        if (!validId.equals(id))
            verify = false;

        // Recreate message
        String messageToCheck = id + "," + String.valueOf(e) + String.valueOf(n) + "," + operation + "," + value;
```

```java

        // Verify if signature is valid
        if (!sign(messageToCheck, sign))
            verify = false;

        if (verify)
            userId = validId;

        return verify;
    }

    // Method to convert bytes to hex
    // Reference Code: https://mkyong.com/java/java-how-to-convert-bytes-to-hex/
    1 usage
    public static String bytesToHex(byte[] bytes) {
        StringBuilder result = new StringBuilder();

        // Step 1: Iterate through each byte in the 'bytes' array.
        for (byte aByte : bytes) {

            // Step 2: Convert the byte to an integer and apply a bitwise AND operation with 0xff.
            int decimal = (int) aByte & 0xff; // This ensures that the integer is non-negative.

            // Step 3: Convert the decimal value to a hexadecimal string.
            String hex = Integer.toHexString(decimal);

            // Step 4: Check if the hex string has an odd length, and if so, pad it with a leading '0'.
            if (hex.length() % 2 == 1) {
                hex = "0" + hex;
            }

            // Step 5: Append the hexadecimal value to the result.
            result.append(hex);
        }

        // Step 6: Return the concatenated hexadecimal string representation of the bytes.
        return result.toString();
    }

    // Method to validate the user id
    1 usage
    public static String validateId(String key) throws NoSuchAlgorithmException {
        // Step 1: Create a MessageDigest instance using SHA-256 hashing algorithm.
        MessageDigest digest = MessageDigest.getInstance( algorithm: "SHA-256");

        // Step 2: Generate a hash of the input 'key'.
        byte[] hashBytes = digest.digest(key.getBytes());

        // Step 3: Create a new byte array to store the final 20 bytes of the hash.
        byte[] finalBytes = new byte[20];

        // Step 4: Copy the least significant 20 bytes of the hash into the final array.
        // This is done to ensure that the output is a 20-byte (160-bit) hash.
        System.arraycopy(hashBytes,  srcPos: hashBytes.length - 20, finalBytes,  destPos: 0,  length: 20);

        // Step 5: Convert the byte array to a hexadecimal string representation.
        return bytesToHex(finalBytes);
    }
```

```java

    // Method to check whether signed message is valid
    // Reference Code : ShortMessageVerify.java (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
    1 usage
    public static boolean sign(String messageToCheck, String encryptedHashStr) throws
            UnsupportedEncodingException, NoSuchAlgorithmException {

        // Take the encrypted string and make it a big integer
        BigInteger encryptedHash = new BigInteger(encryptedHashStr);

        // Decrypt the encryptedHash to compute a decryptedHash
        BigInteger decryptedHash = encryptedHash.modPow(e, n);

        // Get the bytes from messageToCheck
        byte[] bytesOfMessageToCheck = messageToCheck.getBytes( charsetName: "UTF-8");

        // Hash the messageToCheck using SHA-256 (be sure to handle the extra byte as described in the signing method.)
        MessageDigest md = MessageDigest.getInstance( algorithm: "SHA-256");
        byte[] messageToCheckDigest = md.digest(bytesOfMessageToCheck);
        messageToCheckDigest[0] = 0;

        // Make it a big int
        BigInteger bigIntegerToCheck = new BigInteger(messageToCheckDigest);

        // If this new hash is equal to the decryptedHash, return true else false.
        if (bigIntegerToCheck.compareTo(decryptedHash) == 0) {
            return true;
        }
        else {
            return false;
        }
    }
}

```

Project2Task5ClientConsole



Project2Task5ServerConsole