

95702: Distributed Systems for Information Systems Management

Project 2

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

Task 0:

Project2Task0Client

```
1 // Ariane Correa
2 // ajcorrea
3
4 package com.example.project2task0;
5
6 import java.net.*;
7 import java.io.*;
8 import java.util.Scanner;
9
10 // Reference: EchoClientUDP.java from Couleuris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
11
12 public class EchoClientUDP {
13
14     // args[] gives server hostname and message content
15     public static void main(String args[]) {
16
17         DatagramSocket aSocket = null; // Declare a DatagramSocket for UDP communication.
18
19         try {
20             System.out.println("The UDP client is running.");
21
22             // Define the server host (in this case, localhost).
23             InetAddress aHost = InetAddress.getByName(host: "localhost");
24
25             Scanner sc = new Scanner(System.in);
26             System.out.println("Provide server side port number");
27
28             int serverPort = sc.nextInt(); // Read the server's port number from the user.
29
30             aSocket = new DatagramSocket(); // Create a DatagramSocket for sending and receiving UDP packets.
31             String nextLine;
32             BufferedReader typed = new BufferedReader(new InputStreamReader(System.in));
33
34             while ((nextLine = typed.readLine()) != null) {
35
36                 // Create a new DatagramPacket to send the message to the server.
37                 byte[] message = nextLine.getBytes();
38                 DatagramPacket packet = new DatagramPacket(message, message.length, aHost, serverPort);
39
40                 aSocket.send(packet); // Send the message to the server.
41
42                 // Create a new DatagramPacket to receive the echo message from the server.
43                 DatagramPacket echo = aSocket.receive();
44
45                 // Print the echo message back to the user.
46                 System.out.println("Echoed message: " + new String(echo.getData()));
47
48             }
49
50         } catch (IOException e) {
51             e.printStackTrace();
52         }
53     }
54 }
```

Project2Task0Client (continued)

```
© EchoClientUDP.java x © EchoServerUDP.java

36         // Create a DatagramPacket for sending data to the server.
37         DatagramPacket request = new DatagramPacket(m, m.length, aHost, serverPort);
38         aSocket.send(request); // Send the data to the server.
39
40         // Initialize a new buffer that will store contents of the reply message
41         byte[] buffer = new byte[1000];
42         // Create a DatagramPacket for receiving data from the server.
43         DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
44         aSocket.receive(reply); // Receive data from the server.
45
46         // We create a new replyArray of request length
47         byte[] replyArray = new byte[request.getLength()];
48         // Copy relevant reply data into the new array based on request length
49         System.arraycopy(reply.getData(), srcPos: 0, replyArray, destPos: 0, reply.getLength());
50         String requestString = new String(replyArray); // Convert received bytes to a string.
51         System.out.println("Reply: " + requestString); // Print the server's reply.
52
53         // Exit if "halt!" is received
54         if (requestString.equalsIgnoreCase(anotherString: "halt!")) {
55             System.out.println("UDP Client side quitting");
56             break;
57         }
58     }
59 }
60 catch (SocketException e) {
61     System.out.println("Socket: " + e.getMessage()); // Handle SocketException if it occurs.
62 }
63 catch (IOException e) {
64     System.out.println("IO: " + e.getMessage()); // Handle IOException if it occurs.
65 }
66 finally {
67     if (aSocket != null) aSocket.close(); // Close the DatagramSocket when done.
68 }
69 }
70 }
```

Project2Task0Server

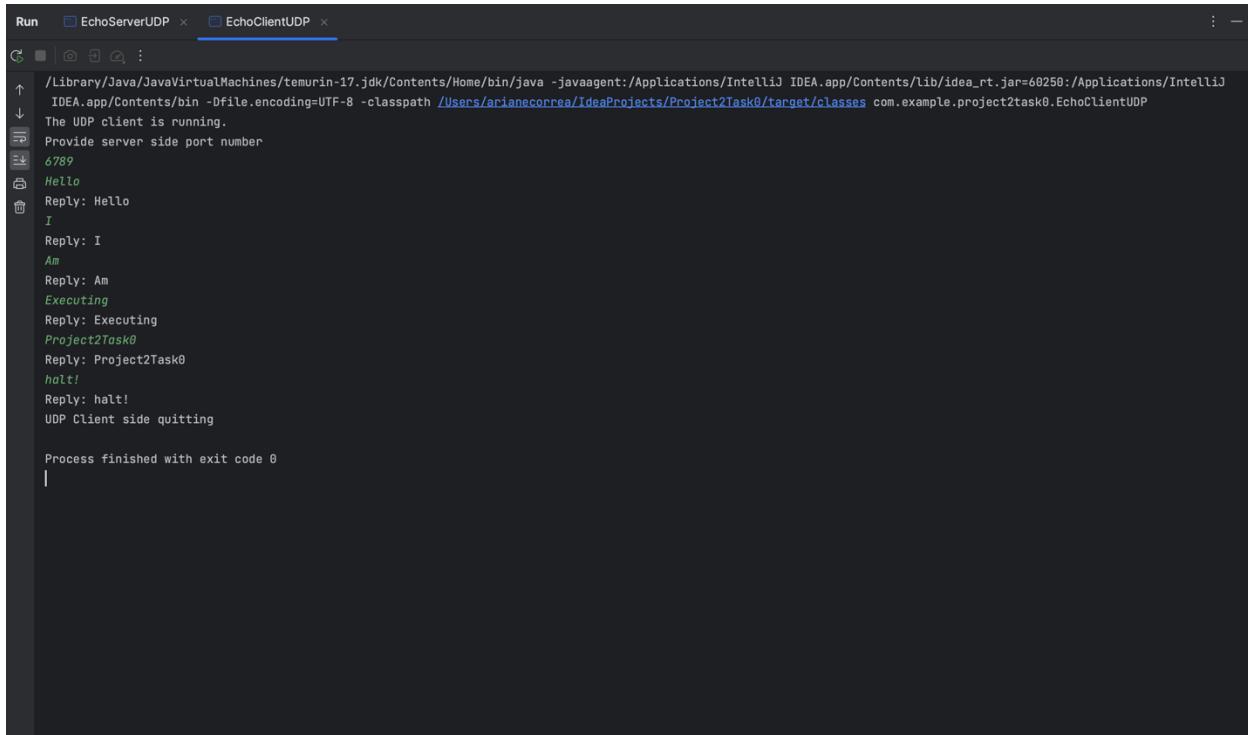
```
© EchoClientUDP.java    © EchoServerUDP.java ×
1 // Ariane Correa
2 // ajcorrea
3 |
4 package com.example.project2task0;
5
6 import java.net.*;
7 import java.io.*;
8 import java.util.Arrays;
9 import java.util.Scanner;
10
11 // Reference: EchoServerUDP.java from Coulonnis text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
12
13 public class EchoServerUDP {
14     public static void main(String args[]) {
15
16         System.out.println("The UDP server is running.");
17
18         DatagramSocket aSocket = null; // Declare a DatagramSocket for UDP communication.
19
20         // We declare a byte array that will receive client messages
21         byte[] buffer = new byte[1000];
22         try {
23
24             Scanner sc = new Scanner(System.in);
25             System.out.println("Provide port number server should listen on");
26             int serverPort = sc.nextInt(); // Read the port number from the user.
27             aSocket = new DatagramSocket(serverPort); // Create a DatagramSocket bound to the specified port.
28             // Create a new datagram packet to fetch the request
29             DatagramPacket request = new DatagramPacket(buffer, buffer.length);
30             while (true) {
31
32                 aSocket.receive(request); // Receive a UDP packet and store it in 'request'.
33
34                 byte[] replyArray = new byte[request.getLength()];
35             }
36         } catch (IOException e) {
37             e.printStackTrace();
38         }
39     }
40 }
```

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

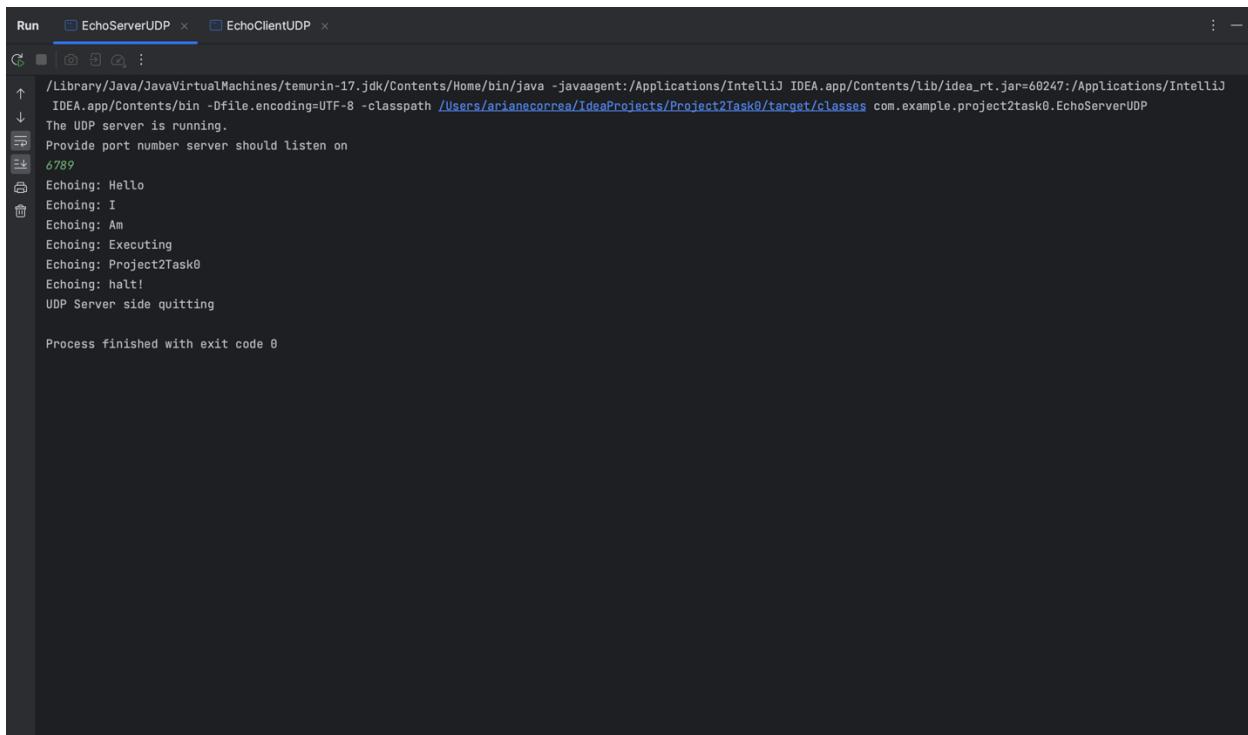


```
Run EchoServerUDP × EchoClientUDP ×

/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=60250:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task0/target/classes com.example.project2task0.EchoClientUDP
The UDP client is running.
Provide server side port number
6789
Hello
Reply: Hello
I
Reply: I
Am
Reply: Am
Executing
Reply: Executing
Project2Task0
Reply: Project2Task0
halt!
Reply: halt!
UDP Client side quitting

Process finished with exit code 0
```

Project2Task0ServerConsole



```
Run EchoServerUDP × EchoClientUDP ×

/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=60247:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task0/target/classes com.example.project2task0.EchoServerUDP
The UDP server is running.
Provide port number server should listen on
6789
Echoing: Hello
Echoing: I
Echoing: Am
Echoing: Executing
Echoing: Project2Task0
Echoing: halt!
UDP Server side quitting

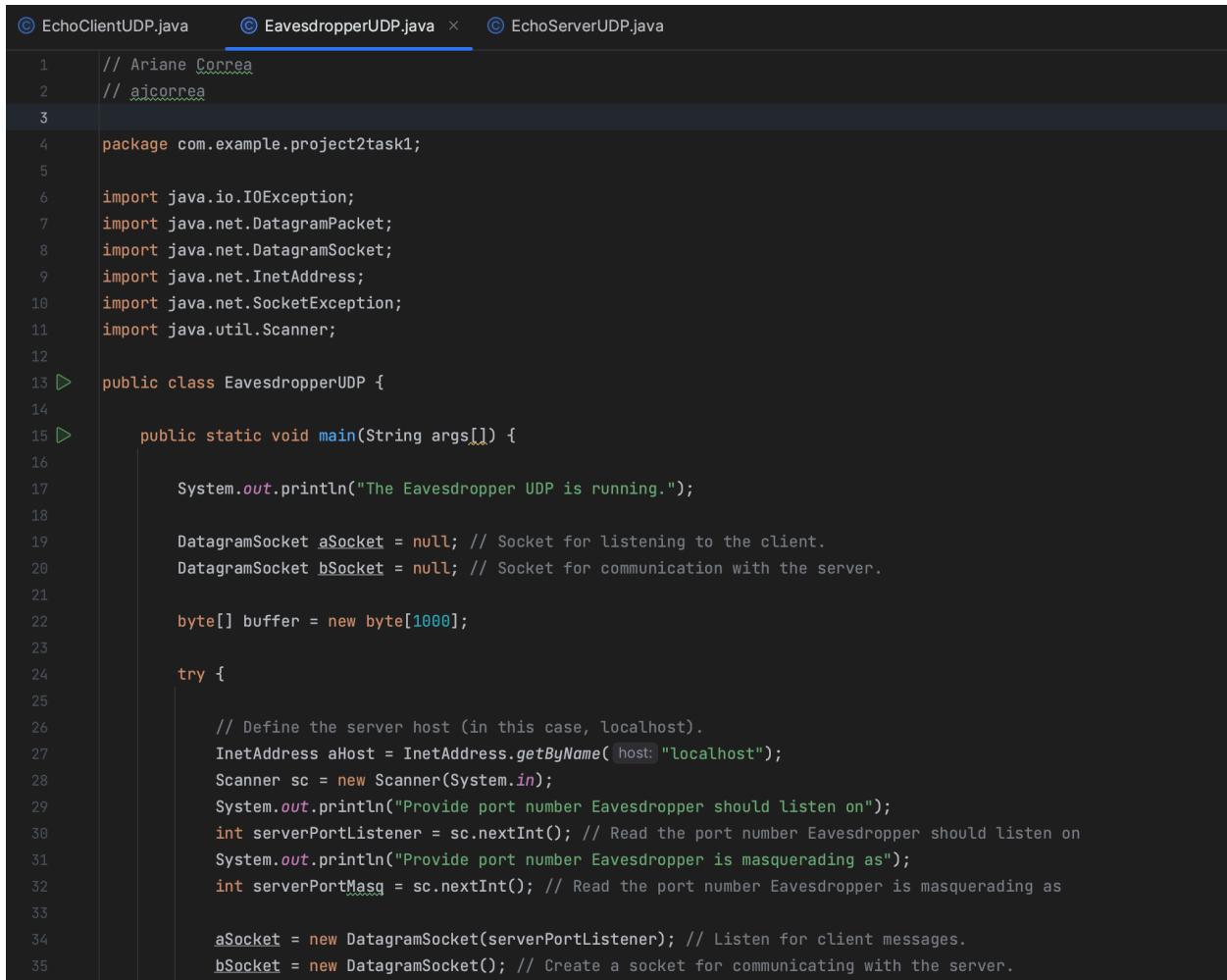
Process finished with exit code 0
```

Project 2

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

Task 1:

EavesdropperUDP.java



The screenshot shows a code editor with three tabs: EchoClientUDP.java, EavesdropperUDP.java (which is selected), and EchoServerUDP.java. The EavesdropperUDP.java tab contains the following Java code:

```
1 // Ariane Correa
2 // ajcorrea
3
4 package com.example.project2task1;
5
6 import java.io.IOException;
7 import java.net.DatagramPacket;
8 import java.net.DatagramSocket;
9 import java.net.InetAddress;
10 import java.net.SocketException;
11 import java.util.Scanner;
12
13 public class EavesdropperUDP {
14
15     public static void main(String args[]) {
16
17         System.out.println("The Eavesdropper UDP is running.");
18
19         DatagramSocket aSocket = null; // Socket for listening to the client.
20         DatagramSocket bSocket = null; // Socket for communication with the server.
21
22         byte[] buffer = new byte[1000];
23
24         try {
25
26             // Define the server host (in this case, localhost).
27             InetAddress aHost = InetAddress.getByName("localhost");
28             Scanner sc = new Scanner(System.in);
29             System.out.println("Provide port number Eavesdropper should listen on");
30             int serverPortListener = sc.nextInt(); // Read the port number Eavesdropper should listen on
31             System.out.println("Provide port number Eavesdropper is masquerading as");
32             int serverPortMasq = sc.nextInt(); // Read the port number Eavesdropper is masquerading as
33
34             aSocket = new DatagramSocket(serverPortListener); // Listen for client messages.
35             bSocket = new DatagramSocket(); // Create a socket for communicating with the server.
36
37         } catch (IOException e) {
38             e.printStackTrace();
39         }
40
41     }
42
43 }
```

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)

```
© EchoClientUDP.java © EavesdropperUDP.java × © EchoServerUDP.java

71
72         aSocket.send(reply); // Send the server's reply to the client.
73
74     }
75
76 }
77 catch (SocketException e) {
78     System.out.println("Socket: " + e.getMessage()); // Handle SocketException if it occurs.
79 }
80 catch (IOException e) {
81     System.out.println("IO: " + e.getMessage()); // Handle IOException if it occurs.
82 }
83 finally {
84     // Close the DatagramSockets when done.
85     if (aSocket != null) aSocket.close();
86     if (bSocket != null) bSocket.close();
87 }
88 }
89
90 // Function to check if a string contains a whole word.
91 @usage
92 private static boolean containsWholeWord(String input, String word) {
93     String regex = "\\b" + word + "\\b";
94     return input.matches(regex);
95 }
96
97 // Function to replace a whole word in a string.
98 @usage
99 private static String replaceWholeWord(String input, String wordToReplace, String replacement) {
100     String regex = "\\b" + wordToReplace + "\\b";
101     return input.replaceAll(regex, replacement);
102 }
```

Project2Task1ThreeConsoles

Case 1: Correct Server

Client Console

```
Run EchoServerUDP × EchoClientUDP × EavesdropperUDP ×
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=61094:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task1/target/classes com.example.project2task1.EchoClientUDP
↓ The UDP client is running.
Provide server side port number
6789
Running
Reply: Running
Project2Task1
Reply: Project2Task1
Now
Reply: Now
halt!
Reply: halt!
UDP Client side quitting

Process finished with exit code 0
```

Server Console

```
Run EchoServerUDP × EchoClientUDP × EavesdropperUDP ×
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=61091:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task1/target/classes com.example.project2task1.EchoServerUDP
↓ The UDP server is running.
Provide port number server should listen on
6789
Echoing: Running
Echoing: Project2Task1
Echoing: Now
Echoing: halt!
UDP Server side quitting

Process finished with exit code 0
```

Eavesdropper Console

```
Run EchoServerUDP × EchoClientUDP × EavesdropperUDP ×
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=61100:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task1/target/classes com.example.project2task1.EavesdropperUDP
↓ The Eavesdropper UDP is running.
Provide port number Eavesdropper should listen on
6798
Provide port number Eavesdropper is masquerading as
6789
```

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

```
Run EchoServerUDP × EchoClientUDP × EavesdropperUDP ×
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=61109:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task1/target/classes com.example.project2task1.EchoClientUDP
The UDP client is running.
Provide server side port number
6798
I
Reply: I
dislike
Reply: dislike
running
Reply: running
Project2Task1
Reply: Project2Task1
halt!
Reply: halt!
UDP Client side quitting

Process finished with exit code 0
```

Server Console

```
Run EchoServerUDP × EchoClientUDP × EavesdropperUDP ×
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=61106:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task1/target/classes com.example.project2task1.EchoServerUDP
The UDP server is running.
Provide port number server should listen on
6789
Echoing: I
Echoing: dislike
Echoing: running
Echoing: Project2Task1
Echoing: halt!
UDP Server side quitting

Process finished with exit code 0
```

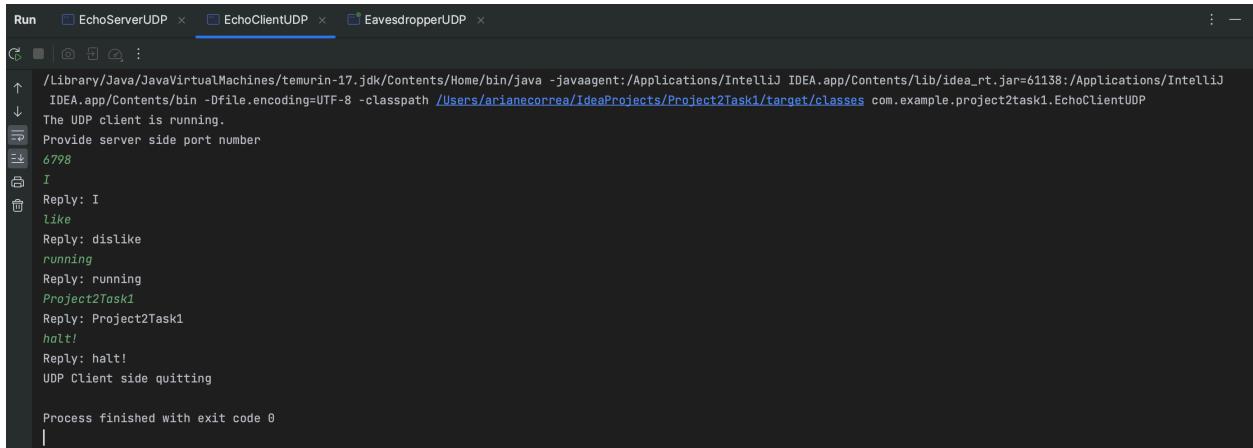
Eavesdropper Console

```
Run EchoServerUDP × EchoClientUDP × EavesdropperUDP ×
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=61112:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task1/target/classes com.example.project2task1.EavesdropperUDP
The Eavesdropper UDP is running.
Provide port number Eavesdropper should listen on
6798
Provide port number Eavesdropper is masquerading as
6789
Client Message: I
Server Reply: I
Client Message: dislike
Server Reply: dislike
Client Message: running
Server Reply: running
Client Message: Project2Task1
Server Reply: Project2Task1
Client Message: halt!
Server Reply: halt!
```

Project2Task1ThreeConsoles

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

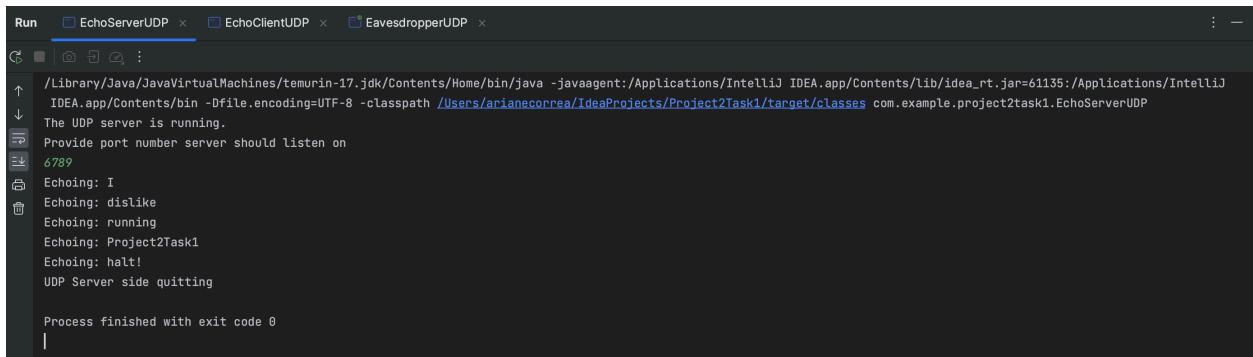
Client Console



```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=61138:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task1/target/classes com.example.project2task1.EchoClientUDP
The UDP client is running.
Provide server side port number
6798
I
Reply: I
like
Reply: dislike
running
Reply: running
Project2Task1
Reply: Project2Task1
halt!
Reply: halt!
UDP Client side quitting

Process finished with exit code 0
```

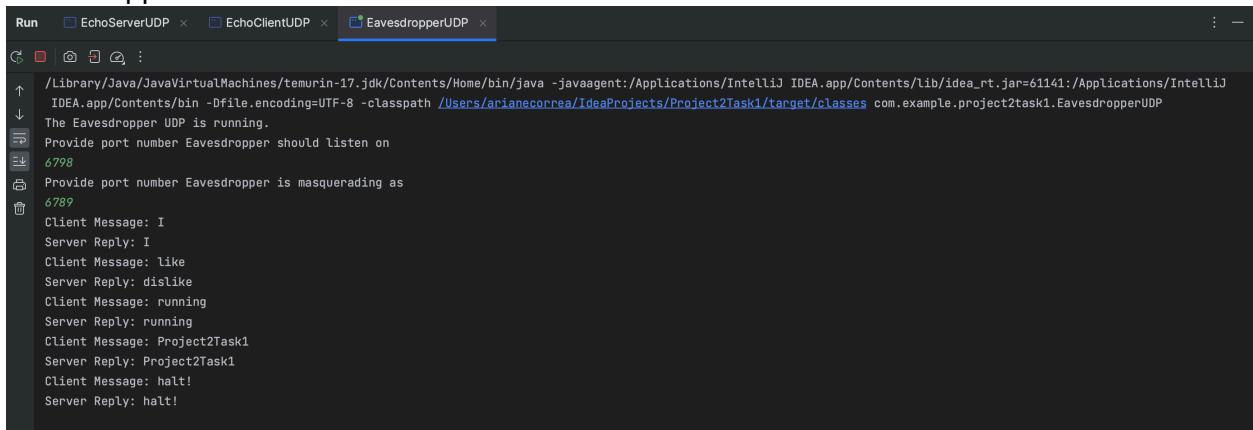
Server Console



```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=61135:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task1/target/classes com.example.project2task1.EchoServerUDP
The UDP server is running.
Provide port number server should listen on
6789
Echoing: I
Echoing: dislike
Echoing: running
Echoing: Project2Task1
Echoing: halt!
UDP Server side quitting

Process finished with exit code 0
```

Eavesdropper Console



```
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=61141:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task1/target/classes com.example.project2task1.EavesdropperUDP
The Eavesdropper UDP is running.
Provide port number Eavesdropper should listen on
6798
Provide port number Eavesdropper is masquerading as
6789
Client Message: I
Server Reply: I
Client Message: like
Server Reply: dislike
Client Message: running
Server Reply: running
Client Message: Project2Task1
Server Reply: Project2Task1
Client Message: halt!
Server Reply: halt!
```

Project 2

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

Task 2:

Project2Task2Client

```
© AddingClientUDP.java × © AddingServerUDP.java
1 // Ariane Correa
2 // ajcorrea
3
4 package com.example.project2task2;
5
6 import java.io.BufferedReader;
7 import java.io.IOException;
8 import java.io.InputStreamReader;
9 import java.net.DatagramPacket;
10 import java.net.DatagramSocket;
11 import java.net.InetAddress;
12 import java.net.SocketException;
13 import java.nio.ByteBuffer;
14 import java.util.Scanner;
15
16 // Reference: EchoClientUDP.java from Couleuris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
17
18 ▷ public class AddingClientUDP {
19
20     2 usages
21     static int serverPort;
22     5 usages
23     static DatagramSocket aSocket = null;
24
25     // args[] gives server hostname and message content
26     public static void main(String args[]) {
27
28         try {
29             System.out.println("The UDP client is running.");
30
31             Scanner sc = new Scanner(System.in);
32             System.out.println("Provide server side port number");
33             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("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
63     1 usage
64     public static int add(int i) throws IOException {
65
66         // Define the server host (in this case, localhost).
67         InetAddress aHost = InetAddress.getByName("localhost");
68
69         // Convert user input to bytes to send via UDP
70         byte[] m = String.valueOf(i).getBytes();
71
72         // Create a DatagramPacket for sending data to the server.
73         DatagramPacket request = new DatagramPacket(m, m.length, aHost, serverPort);
74         aSocket.send(request); // Send the data to the server.
75
76         // Initialize a new buffer that will store contents of the reply message
77         byte[] buffer = new byte[1000];
78
79         // Create a DatagramPacket for receiving data from the server.
80         DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
81         aSocket.receive(reply); // Receive data from the server.
82
83         // Use reply.getLength() to determine the actual length of the received message
84         byte[] replyArray = new byte[reply.getLength()];
85
86         // Copy relevant reply data into the new array based on reply length
87         System.arraycopy(reply.getData(), 0, replyArray, 0, reply.getLength());
88
89         // Convert the byte array back to an integer using ByteBuffer
90         // Reference Code: https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/nio/ByteBuffer.html
91         int n = ByteBuffer.wrap(replyArray).getInt();
92
93     }
94 }
```

Project2Task2Server

```
© AddingClientUDP.java      © AddingServerUDP.java ×
1 // Ariane Correa
2 // ajcorrea
3
4 package com.example.project2task2;
5
6 import java.io.IOException;
7 import java.math.BigInteger;
8 import java.net.DatagramPacket;
9 import java.net.DatagramSocket;
10 import java.net.SocketException;
11 import java.nio.ByteBuffer;
12 import java.util.Scanner;
13
14 // Reference: EchoServerUDP.java from Couloris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
15
16 public class AddingServerUDP {
17     3 usages
18     static int i = 0;
19
20     public static void main(String args[]) {
21
22         System.out.println("The UDP server is running.");
23
24         DatagramSocket aSocket = null; // Declare a DatagramSocket for UDP communication.
25
26         // We declare a byte array that will receive client messages
27         byte[] buffer = new byte[1000];
28
29         try {
30             Scanner sc = new Scanner(System.in);
31             System.out.println("Provide port number server should listen on");
32             int serverPort = sc.nextInt(); // Read the port number from the user.
33             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
74 // usage
75 public static int add(int n) {
76     i += n;
77     return i;
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

```
Run AddingServerUDP × AddingClientUDP ×
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=61902:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task2/target/classes com.example.project2task2.AddingClientUDP
The UDP Client is running.
Provide server side port number
6789
1
The server returned 1.
2
The server returned 3.
-3
The server returned 0.
4
The server returned 4.
5
The server returned 9.
halt!
UDP Client side quitting

Process finished with exit code 0
```

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

```
Run AddingServerUDP × AddingClientUDP ×
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=61907:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task2/target/classes com.example.project2task2.AddingClientUDP
The UDP Client is running.
Provide server side port number
6789
6
The server returned 15.
7
The server returned 22.
-8
The server returned 14.
9
The server returned 23.
10
The server returned 33.
halt!
UDP Client side quitting

Process finished with exit code 0 |
```

Project2Task2ServerConsole

```
Run AddingServerUDP x AddingClientUDP x

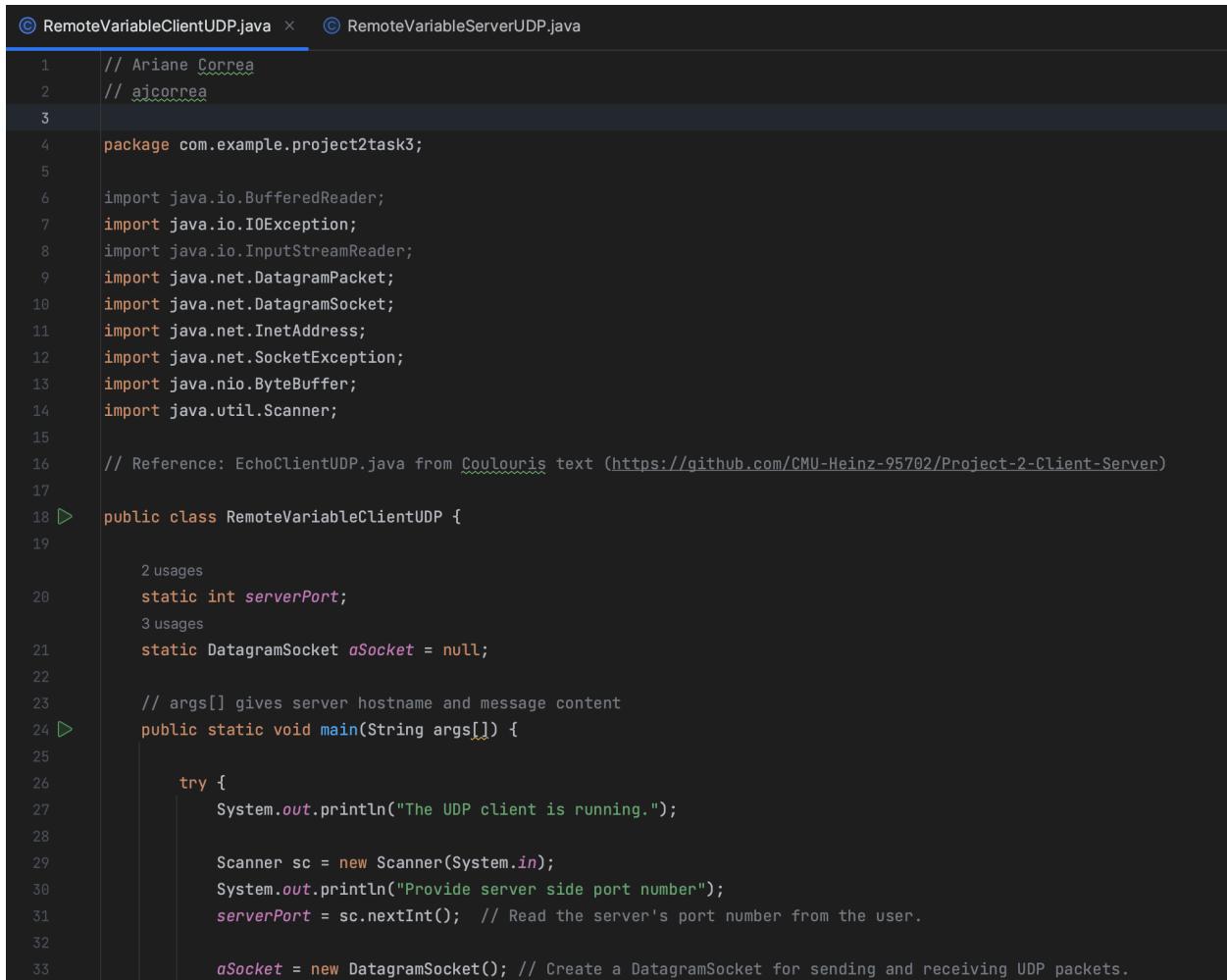
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=61899:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task2/target/classes com.example.project2task2.AddingServerUDP
The UDP server is running.
Provide port number server should listen on
6789
Adding 1 to 0
Returning sum of 1 to client
Adding 2 to 1
Returning sum of 3 to client
Adding -3 to 3
Returning sum of 0 to client
Adding 4 to 0
Returning sum of 4 to client
Adding 5 to 4
Returning sum of 9 to client
Adding 6 to 9
Returning sum of 15 to client
Adding 7 to 15
Returning sum of 22 to client
Adding -8 to 22
Returning sum of 14 to client
Adding 9 to 14
Returning sum of 23 to client
Adding 10 to 23
Returning sum of 33 to client
```

Project 2

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

Task 3:

Project2Task3Client



```
© RemoteVariableClientUDP.java × © RemoteVariableServerUDP.java
1 // Ariane Correa
2 // ajcorrea
3
4 package com.example.project2task3;
5
6 import java.io.BufferedReader;
7 import java.io.IOException;
8 import java.io.InputStreamReader;
9 import java.net.DatagramPacket;
10 import java.net.DatagramSocket;
11 import java.net.InetAddress;
12 import java.net.SocketException;
13 import java.nio.ByteBuffer;
14 import java.util.Scanner;
15
16 // Reference: EchoClientUDP.java from Coulouris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
17
18 public class RemoteVariableClientUDP {
19
20     2 usages
21     static int serverPort;
22     3 usages
23     static DatagramSocket aSocket = null;
24
25     // args[] gives server hostname and message content
26     public static void main(String args[]) {
27
28         try {
29             System.out.println("The UDP client is running.");
30
31             Scanner sc = new Scanner(System.in);
32             System.out.println("Provide server side port number");
33             serverPort = sc.nextInt(); // Read the server's port number from the user.
34
35             aSocket = new DatagramSocket(); // Create a DatagramSocket for sending and receiving UDP packets.
36
37         } catch (IOException e) {
38             e.printStackTrace();
39         }
40     }
41 }
```

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 = "subtract";
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
69     int userId = sc.nextInt();
70
71     String message = userId + "," + op + "," + value;
72
73     if (message != null) {
74         int x = result(message);
75         System.out.println("The server returned " + x);
76     }
77
78 }
79
80 catch (SocketException e) {
81     System.out.println("Socket: " + e.getMessage()); // Handle SocketException if it occurs.
82 }
83
84 catch (IOException e) {
85     System.out.println("IO: " + e.getMessage()); // Handle IOException if it occurs.
86 }
87
88 // Method that returns the input from the server
89 public static int result(String x) throws IOException {
90
91     // Define the server host (in this case, localhost).
92     InetAddress aHost = InetAddress.getByName( host: "localhost");
93
94     // Convert user input to bytes to send via UDP
95     byte[] m = String.valueOf(x).getBytes();
96
97     // Create a DatagramPacket for sending data to the server.
98     DatagramPacket request = new DatagramPacket(m, m.length, aHost, serverPort);
99     aSocket.send(request); // Send the data to the server.
100
101    // Initialize a new buffer that will store contents of the reply message
102    byte[] buffer = new byte[1000];
103
104    // Create a DatagramPacket for receiving data from the server.
105    DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
106    aSocket.receive(reply); // Receive data from the server.
107
108    // Use reply.getLength() to determine the actual length of the received message
109    byte[] replyArray = new byte[reply.getLength()];
110
111    // Copy relevant reply data into the new array based on reply length
112    System.arraycopy(reply.getData(), [srcPos: 0, replyArray, [destPos: 0, reply.getLength()]);
113
114    // Convert the byte array back to an integer using ByteBuffer
115    // Reference Code: https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/nio/ByteBuffer.html
116    int n = ByteBuffer.wrap(replyArray).getInt();
117
118    return n;
119 }
120
121 }
```

Project2Task3Server

```
② RemoteVariableClientUDP.java ② RemoteVariableServerUDP.java ×
1 // Ariane Correa
2 // ejcorrea
3 |
4 package com.example.project2task3;
5
6 import java.io.IOException;
7 import java.net.DatagramPacket;
8 import java.net.DatagramSocket;
9 import java.net.SocketException;
10 import java.nio.ByteBuffer;
11 import java.util.HashMap;
12 import java.util.Map;
13 import java.util.Scanner;
14
15 // Reference: EchoServerUDP.java from Couleuris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
16
17 ▶ public class RemoteVariableServerUDP {
18
19     //static int i = 0;
20
21     // Map to store user IDs and their corresponding values
22     // 11 usages
23     static Map<Integer, Integer> UserID_Value_Map = new HashMap<>();
24
25     ▶ public static void main(String args[]) {
26
27         System.out.println("The UDP server is running.");
28
29         DatagramSocket aSocket = null; // Declare a DatagramSocket for UDP communication.
30
31         // We declare a byte array that will receive client messages
32         byte[] buffer = new byte[1000];
33
34         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     while (true) {

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

43         byte[] replyArray = new byte[request.getLength()];
44         // Copy the message to the reply array which is the length of the request
45         System.arraycopy(request.getData(), srcPos: 0, replyArray, destPos: 0, request.getLength());
46         String requestString = new String(replyArray);

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

51         byte[] byteArray;
52         System.out.println("User " + userId + " requested " + operation + " operation.");

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

55         // If the operation selected is Addition
56         if (operation.equalsIgnoreCase( anotherString: "add" )) {
57             n = add(Integer.parseInt(userId), Integer.parseInt(value));
58         }
59         // If the operation selected is Subtraction
60         else if (operation.equalsIgnoreCase( anotherString: "subtract" )) {
61             n = subtract(Integer.parseInt(userId), Integer.parseInt(value));
62         }
63     }
64 }
```

Project2Task3Server (continued)

```
© RemoteVariableClientUDP.java      © RemoteVariableServerUDP.java ×

70
71     // If the operation selected is Get
72     else if (operation.equalsIgnoreCase( anotherString: "get")) {
73         n = get(Integer.parseInt(userId));
74     }
75
76     // Print to Console
77     System.out.println("Returning " + n + " to client");
78
79     // Convert the value of ByteBuffer to return back to Client
80     // Reference Code: https://docs.oracle.com/en/java/javase/14/docs/api/java.base/java/nio/ByteBuffer.html
81     byteArray = ByteBuffer.allocate( capacity: 4).putInt(n).array();
82
83     // Create a new datagram packet with the newly created reply array, request hosts and port array
84     DatagramPacket reply = new DatagramPacket(byteArray,
85                                              byteArray.length, request.getAddress(), request.getPort());
86     aSocket.send(reply); // Send a reply back to the client.
87
88     // If the user enters halt!, Close the Server Socket
89     if (requestString.equalsIgnoreCase( anotherString: "halt!")) {
90         System.out.println("UDP Server side quitting");
91         break;
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
106     1 usage
107     public static int add(int userId, int n) {
108         int result;
109         if (UserID_Value_Map.containsKey(userId)) {
110             result = UserID_Value_Map.get(userId) + n;
111             UserID_Value_Map.put(userId, result);
112         }
113         else {
114             result = n;
115             UserID_Value_Map.put(userId, n);
116         }
117         return result;
118     }
119
120     // Subtracts value received from the specific user
121     1 usage
122     public static int subtract(int userId, int n) {
123         int result;
124         if (UserID_Value_Map.containsKey(userId)) {
125             result = UserID_Value_Map.get(userId) - n;
126             UserID_Value_Map.put(userId, result);
127         }
128         else {
129             result = n;
130             UserID_Value_Map.put(userId, n);
131         }
132         return result;
133     }
134
135     // Gets value received from the specific user
136     1 usage
137     public static int get(int userId) {
138         int result;
139         if (UserID_Value_Map.containsKey(userId)) {
140             result = UserID_Value_Map.get(userId);
141         }
142         else {
143             result = 0;
144             UserID_Value_Map.put(userId, 0);
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

```
Run  RemoteVariableServerUDP  ×  RemoteVariableClientUDP  ×

↑ /Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=62339:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorreia/IdeaProjects/Project2Task3/target/classes com.example.project2task3.RemoteVariableClientUDP
↓ The UDP client is running.
Provide server side port number
6789
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:
5
Enter your ID:
101
The server returned 5
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:
10
Enter your ID:
102
The server returned 10
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:
15
Enter your ID:
103
The server returned 15
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:
2
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
2
Enter value to subtract:
4
Enter your ID:
102
The server returned 6
```

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

```
Run  RemoteVariableServerUDP  RemoteVariableClientUDP

/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=62383:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianeocrea/IdeaProjects/Project2Task3/target/classes com.example.project2task3.RemoteVariableClientUDP

The UDP client is running.
Provide server side port number
6789
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
```

Project2Task3ServerConsole

```
Run  RemoteVariableServerUDP  RemoteVariableClientUDP  : 

/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=62336:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorreia/IdeaProjects/Project2Task3/target/classes com.example.project2task3.RemoteVariableServerUDP
The UDP server is running.
Provide port number server should listen on
6789
User 101 requested add operation.
Returning 5 to client
User 102 requested add operation.
Returning 10 to client
User 103 requested add operation.
Returning 15 to client
User 101 requested subtract operation.
Returning 3 to client
User 102 requested subtract operation.
Returning 6 to client
User 103 requested subtract operation.
Returning 9 to client
User 101 requested get operation.
Returning 3 to client
User 102 requested get operation.
Returning 6 to client
User 103 requested get operation.
Returning 9 to client
User 101 requested get operation.
Returning 3 to client
User 102 requested get operation.
Returning 6 to client
User 103 requested get operation.
Returning 9 to client
```

Project 2

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

Task 4:

Project2Task4Client

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

public class RemoteVariableClientTCP {

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

    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);
        }
    }
}
```

```

30     while (!clientSocket.isClosed()) {
31
32         // Display Menu of Operation Options to Client Console
33         System.out.println("1. Add a value to your sum.\n" +
34             "2. Subtract a value from your sum.\n" +
35             "3. Get your sum.\n" +
36             "4. Exit client");
37
38         int option = sc.nextInt();
39         int value = 0;
40         String op = "";
41
42         // Check the user's choice and set the operation and value accordingly
43         if (option == 1) {
44             System.out.println("Enter value to add:");
45             value = sc.nextInt();
46             op = "add";
47         }
48
49         if (option == 2) {
50             System.out.println("Enter value to subtract:");
51             value = sc.nextInt();
52             op = "subtract";
53         }
54
55         if (option == 3)
56             op = "get";
57
58         if (option == 4) {
59             System.out.println("Client side quitting. The remote variable server is still running.");
60             break;
61         }
62
63         System.out.println("Enter your ID:");
64         int userId = sc.nextInt();
65
66         String message = userId + "," + op + "," + value;
67
68         if (message != null) {
69             int x = result(message);
70             System.out.println("The server returned " + x);
71         }
72     }
73
74     catch (IOException e) {
75         // Handle IO Exception by displaying an error message
76         System.out.println("IO Exception: " + e.getMessage());
77     }
78     finally {
79         try {
80             // Ensure that the clientSocket is closed, if it's not null
81             if (clientSocket != null) {
82                 clientSocket.close();
83             }
84         }
85         catch (IOException e) {
86             // Handle any potential IOException that may occur when closing the socket
87             // This is typically ignored, as the main goal is to ensure the socket is closed
88         }
89     }
90 }

```

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

Project2Task4Server

```
© RemoteVariableServerTCP.java × © RemoteVariableClientTCP.java
1 // Ariane Correa
2 // aicorrea
3
4 package com.example.project2task4;
5
6 import java.io.BufferedWriter;
7 import java.io.IOException;
8 import java.io.OutputStreamWriter;
9 import java.io.PrintWriter;
10 import java.net.*;
11 import java.nio.ByteBuffer;
12 import java.util.HashMap;
13 import java.util.Map;
14 import java.util.Scanner;
15 import java.util.TreeMap;
16
17 // Reference: EchoServerUDP.java from Couleuris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
18 // Reference: EchoServerTCP.java from Couleuris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
19
20 public class RemoteVariableServerTCP {
21
22     // Map to store user IDs and their corresponding values
23     static Map<Integer, Integer> UserID_Value_Map = new TreeMap<>();
24
25     public static void main(String args[]) {
26
27         Socket clientSocket = null; // Declare and Initialize Client Socket
28         System.out.println("The TCP server is running.");
29
30         try {
31
32             Scanner sc = new Scanner(System.in);
33             System.out.println("Provide port number server should listen on");
34             int serverPort = sc.nextInt(); // Read the port number from the user.
35             ServerSocket listenSocket = new ServerSocket(serverPort); //Initialize Server Socket
```

```
36
37     /*
38      * Block waiting for a new connection request from a client.
39      * When the request is received, "accept" it, and the rest
40      * the tcp protocol handshake will then take place, making
41      * the socket ready for reading and writing.
42      */
43     clientSocket = listenSocket.accept();
44     // If we get here, then we are now connected to a client.
45
46     // Set up "in" to read from the client socket
47     Scanner in;
48     in = new Scanner(clientSocket.getInputStream());
49
50     // Set up "out" to write to the client socket
51     PrintWriter out;
52     out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));
53
54     /*
55      * Forever,
56      *   read a line from the socket
57      *   print it to the console
58      *   echo it (i.e. write it) back to the client
59      */
60
61     while (!clientSocket.isClosed()) {
62
63         // Check if there is a line available to be read
64         // If not, it is because Client side has quit, but server can continue running
65         if (in.hasNextLine()) {
66             String requestString = in.nextLine();
67
68             // Use the delimiter to fetch different values
69             String userId = requestString.split( regex: ",") [0];
70             String operation = requestString.split( regex: ",") [1];
71             String value = requestString.split( regex: ",") [2];
72
73             System.out.println("User " + userId + " requested " + operation + " operation.");
74
75             // Initialize integer variable that will store the updated value
76             // after the operation is performed on value
77             int n = 0;
78
79             // If the operation selected is Addition
80             if (operation.equalsIgnoreCase( anotherString: "add")) {
81                 n = add(Integer.parseInt(userId), Integer.parseInt(value));
82             }
83             // If the operation selected is Subtraction
84             else if (operation.equalsIgnoreCase( anotherString: "subtract")) {
85                 n = subtract(Integer.parseInt(userId), Integer.parseInt(value));
86             }
87             // If the operation selected is Get
88             else if (operation.equalsIgnoreCase( anotherString: "get")) {
89                 n = get(Integer.parseInt(userId));
90             }
91
92     }
```

```
91         // Print to Console
92         System.out.println("Returning " + n + " to client");
93         out.println(n);
94         out.flush();
95     }
96 }
97 }
98 }
99 catch (IOException e) {
100     // Handle IO Exception by displaying an error message
101     System.out.println("IO Exception: " + e.getMessage());
102 }
103 finally {
104     try {
105         // Ensure that the clientSocket is closed, if it's not null
106         if (clientSocket != null) {
107             clientSocket.close();
108         }
109     }
110     catch (IOException e) {
111         // Handle any potential IOException that may occur when closing the socket
112         // This is typically ignored, as the main goal is to ensure the socket is closed
113     }
114 }
115 }
116
117 // Adds value received to the specific user
118 public static int add(int userId, int n) {
119     int result;
120     if (UserID_Value_Map.containsKey(userId)) {
121         result = UserID_Value_Map.get(userId) + n;
122         UserID_Value_Map.put(userId, result);
123     }
124     else {
125         result = n;
126         UserID_Value_Map.put(userId, n);
127     }
128     return result;
129 }
130 }
```

```
131 // Subtracts value received from the specific user
132 1 usage
133 public static int subtract(int userId, int n) {
134     int result;
135     if (UserID_Value_Map.containsKey(userId)) {
136         result = UserID_Value_Map.get(userId) - n;
137         UserID_Value_Map.put(userId, result);
138     }
139     else {
140         result = n;
141         UserID_Value_Map.put(userId, n);
142     }
143     return result;
144 }
145 // Gets value received from the specific user
146 1 usage
147 public static int get(int userId) {
148     int result;
149     if (UserID_Value_Map.containsKey(userId)) {
150         result = UserID_Value_Map.get(userId);
151     }
152     else {
153         result = 0;
154         UserID_Value_Map.put(userId, 0);
155     }
156     return result;
157 }
158 }
```

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

```
Run  RemoteVariableServerTCP ×  RemoteVariableClientTCP × : - 
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=49638:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianecorrea/IdeaProjects/Project2Task4/target/classes com.example.project2task4.RemoteVariableClientTCP
The TCP client is running.
Provide server side port number
6789
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:
100
Enter your ID:
1
Received: 100
The server returned 100
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:
10
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:
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
1
Enter value to add:
200
Enter your ID:
2
Received: 200
The server returned 200
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:
20
Enter your ID:
2
Received: 180
The server returned 180
```

```

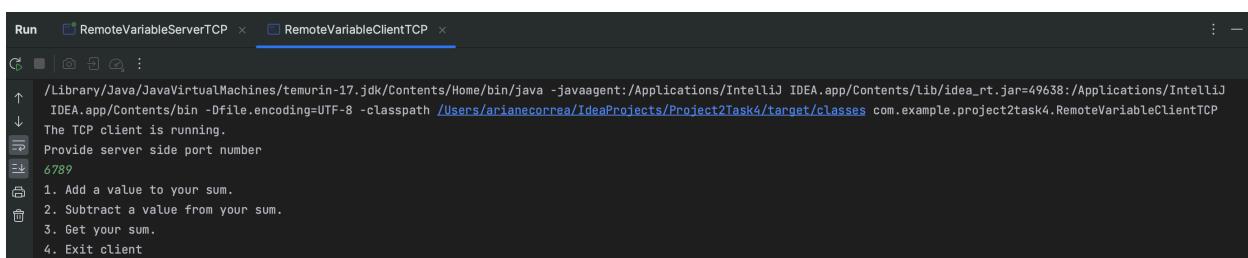
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
2
Received: 180
The server returned 180
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
1
Enter value to add:
300
Enter your ID:
3
Received: 300
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



The screenshot shows the IntelliJ IDEA interface with the 'Run' tool window open. The 'RemoteVariableClientTCP' configuration is selected. The configuration details pane displays the command used to run the server: '/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=49638:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/arianeocrea/IdeaProjects/Project2Task4/target/classes com.example.project2task4.RemoteVariableClientTCP'. Below this, it says 'The TCP client is running.' and 'Provide server side port number' followed by the value '6789'. The 'Output' tab of the tool window shows the interaction between the client and the server, identical to the previous terminal session.

```
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

```
Run  RemoteVariableServerTCP ×  RemoteVariableClientTCP ×  ⋮ —
⟳ ⟲ ⟳ ⟴ : 
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=49634:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/ariane.correa/IdeaProjects/Project2Task4/target/classes com.example.project2task4.RemoteVariableServerTCP
↑
IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/ariane.correa/IdeaProjects/Project2Task4/target/classes com.example.project2task4.RemoteVariableServerTCP
↓
The TCP server is running.
Provide port number server should listen on
6789
User 1 requested add operation.
Returning 100 to client
User 1 requested subtract operation.
Returning 90 to client
User 1 requested get operation.
Returning 90 to client
User 2 requested add operation.
Returning 200 to client
User 2 requested subtract operation.
Returning 180 to client
User 2 requested get operation.
Returning 180 to client
User 3 requested add operation.
Returning 300 to client
User 3 requested subtract operation.
Returning 270 to client
User 3 requested get operation.
Returning 270 to client
User 1 requested get operation.
Returning 90 to client
User 2 requested get operation.
Returning 180 to client
User 3 requested get operation.
Returning 270 to client
```

Project 2

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

Task 5:

Project2Task5Client

```
© VerifyingServerTCP.java   © SigningClientTCP.java ×
1  // Ariane Correa
2  // ajcorrea
3
4  package com.example.project2task5;
5
6  import java.io.*;
7  import java.math.BigInteger;
8  import java.net.InetAddress;
9  import java.net.Socket;
10 import java.net.SocketException;
11 import java.security.MessageDigest;
12 import java.security.NoSuchAlgorithmException;
13 import java.util.Random;
14 import java.util.Scanner;
15
16 // Reference: EchoServerUDP.java from Couleuris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
17 // Reference: EchoServerTCP.java from Couleuris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
18
19 ▶ public class SigningClientTCP {
20
21     2 usages
22     static int serverPort;
23     6 usages
24     static Socket clientSocket = null; // Declare and Initialize Client Socket object
25     6 usages
26     static BigInteger n; // n is the modulus for both the private and public keys
27     5 usages
28     static BigInteger e; // e is the exponent of the public key
29     3 usages
30     static BigInteger d; // d is the exponent of the private key
31
32     // args[] gives server hostname
33     public static void main(String args[]) {
34
35         System.out.println("The TCP client is running.");
36         Scanner sc = new Scanner(System.in);
37     }
38 }
```

```

32     System.out.println("Provide server side port number");
33
34     try {
35         serverPort = sc.nextInt(); // Read the server's port number from the user.
36         clientSocket = new Socket(InetAddress.getByName( host: "localhost"), serverPort);
37
38         // Generate keys once
39         generateKeys();
40         // Generate the userId once
41         String id = generateId( key: String.valueOf(e) + String.valueOf(n));
42
43         while (!clientSocket.isClosed()) {
44
45             // Display Menu of Operation Options to Client Console
46             System.out.println("1. Add a value to your sum.\n" +
47                 "2. Subtract a value from your sum.\n" +
48                 "3. Get your sum.\n" +
49                 "4. Exit client");
50
51             int option = sc.nextInt();
52             int value = 0;
53             String op = "";
54
55             // Check the user's choice and set the operation and value accordingly
56             if (option == 1) {
57                 System.out.println("Enter value to add:");
58                 value = sc.nextInt();
59                 op = "add";
60             }
61
62             if (option == 2) {
63                 System.out.println("Enter value to subtract:");
64                 value = sc.nextInt();
65                 op = "subtract";
66             }
67
68             if (option == 3)
69                 op = "get";
70
71             if (option == 4) {
72                 System.out.println("Client side quitting. The remote variable server is still running.");
73                 break;
74             }
75
76             //generateKeys();
77             //String id = generateId(String.valueOf(e) + String.valueOf(n));
78
79             String message = id + "," + String.valueOf(e) + String.valueOf(n) + "," + op + "," + value;
80
81             if (message != null) {
82                 String x = result(message);
83                 System.out.println("The server returned " + x);
84             }
85         }
86     }

```

```
86     } catch (SocketException e) {
87         // Handle Socket Exception by displaying an error message
88         System.out.println("Socket: " + e.getMessage());
89     } catch (IOException e) {
90         // Handle IO Exception by displaying an error message
91         System.out.println("IO Exception: " + e.getMessage());
92     } catch (NoSuchAlgorithmException ex) {
93         // Handle Runtime Exception by displaying an error message
94         throw new RuntimeException(ex);
95     } finally {
96         try {
97             // Ensure that the clientSocket is closed, if it's not null
98             if (clientSocket != null) {
99                 clientSocket.close();
100            }
101        } catch (IOException e) {
102            // Handle any potential IOException that may occur when closing the socket
103            // This is typically ignored, as the main goal is to ensure the socket is closed
104        }
105    }
106}
107
108 // Method that returns the input from the server @throws IOException @throws NoSuchAlgorithmException
109
110 /**
111  * usage
112  * public static String result(String n) throws IOException, NoSuchAlgorithmException {
113
114     // Combine the input 'n' with its cryptographic signature
115     String encryptedMsg = n + "," + sign(n);
116
117     // Set up input and output streams for communication
118     BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
119     PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));
120
121     // Send the encrypted message to the server
122     out.println(encryptedMsg);
123     out.flush();
124
125     // Read the server's response
126     String data = in.readLine(); // Reads a line of data from the stream
127     System.out.println("Received: " + data);
128
129     // Return the received data
130     return data;
131 }
```

```

130 // Method to generate public and private keys for user
131 1usage
132 public static void generateKeys() {
133     // Each public and private key consists of an exponent and a modulus
134     // Reference Code: RSAExample.java (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
135     Random rnd = new Random();
136
137     // Step 1: Generate two large random prime numbers, p and q.
138     // We use 400 bits here, but best practice for security is 2048 bits.
139     // Change 400 to 2048, recompile, and run the program again, and you will
140     // notice it takes much longer to do the math with that many bits.
141     BigInteger p = new BigInteger( bitLength: 400, certainty: 100, rnd);
142     BigInteger q = new BigInteger( bitLength: 400, certainty: 100, rnd);
143
144     // Step 2: Compute n by the equation n = p * q.
145     n = p.multiply(q);
146
147     // Step 3: Compute phi(n) = (p-1) * (q-1)
148     BigInteger phi = (p.subtract(BigInteger.ONE)).multiply(q.subtract(BigInteger.ONE));
149
150     // Step 4: Select a small odd integer e that is relatively prime to phi(n).
151     // By convention, the prime 65537 is used as the public exponent.
152     e = new BigInteger( val: "65537");
153
154     // Step 5: Compute d as the multiplicative inverse of e modulo phi(n).
155     d = e.modInverse(phi);
156
157     // Step 6: Output the public key (e, n), which is used for encryption.
158     System.out.println("Public key = " + e + n);
159
160     // Step 7: Output the private key (d, n), which is used for decryption.
161     System.out.println("Private key = " + d + n);
162 }
163
164 // Method that returns the input from the server @throws NoSuchAlgorithmException
165 1usage
166 public static String generateId(String key) throws NoSuchAlgorithmException {
167     // Step 1: Create a MessageDigest instance using SHA-256 hashing algorithm.
168     MessageDigest digest = MessageDigest.getInstance( algorithm: "SHA-256");
169
170     // Step 2: Generate a hash of the input 'key'.
171     byte[] hashBytes = digest.digest(key.getBytes());
172
173     // Step 3: Create a new byte array to store the final 20 bytes of the hash.
174     byte[] finalBytes = new byte[20];
175
176     // Step 4: Copy the least significant 20 bytes of the hash into the final array.
177     // This is done to ensure that the output is a 20-byte (160-bit) hash.
178     System.arraycopy(hashBytes, srcPos: hashBytes.length - 20, finalBytes, destPos: 0, length: 20);
179
180     // Step 5: Convert the byte array to a hexadecimal string representation.
181     return bytesToHex(finalBytes);
182 }
```

```
181
182     // Helper function to convert a byte array to a hexadecimal string
183     // Reference Code: https://mkyong.com/java/java-how-to-convert-bytes-to-hex/
184     @
185         String hexString = new StringBuilder();
186         for (byte b : bytes) {
187             // Convert each byte to a 2-character hexadecimal representation
188             String hex = Integer.toHexString( 0xFF & b);
189             if (hex.length() == 1) {
190                 // Pad with leading zero if necessary
191                 hexString.append('0');
192             }
193             hexString.append(hex);
194         }
195         return hexString.toString();
196     }
197
198     // Method to sign the message sent to client using private key @throws UnsupportedEncodingException
199     // @throws NoSuchAlgorithmException
200     // Reference Code : ShortMessageSign.java (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
201     @
202         public static String sign(String message) throws UnsupportedEncodingException, NoSuchAlgorithmException {
203             // Step 1: Convert the input 'message' to bytes using UTF-8 encoding.
204             byte[] bytesOfMessage = message.getBytes( charsetName: "UTF-8");
205
206             // Step 2: Compute the SHA-256 digest of the message.
207             MessageDigest md = MessageDigest.getInstance( algorithm: "SHA-256");
208             byte[] bigDigest = md.digest(bytesOfMessage);
209
210             // Step 3: Set the most significant byte of the digest to 0 (zero).
211             bigDigest[0] = 0;
212
213             // Step 4: Convert the modified digest to a BigInteger.
214             BigInteger m = new BigInteger(bigDigest);
215
216             // Step 5: Encrypt the digest using the private key (d, n) with modular exponentiation.
217             BigInteger c = m.modPow(d, n);
218
219             // Step 6: Return the encrypted digest as a string in BigInteger format.
220             return c.toString();
221         }
222     }
```

Project2Task5Server

```
© VerifyingServerTCP.java × © SigningClientTCP.java
1 // Ariane Correa
2 // gicorrea
3
4 package com.example.project2task5;
5
6 import java.io.*;
7 import java.math.BigInteger;
8 import java.net.ServerSocket;
9 import java.net.Socket;
10 import java.net.SocketException;
11 import java.security.MessageDigest;
12 import java.security.NoSuchAlgorithmException;
13 import java.util.HashMap;
14 import java.util.Map;
15 import java.util.Scanner;
16 import java.util.TreeMap;
17
18 // Reference: EchoServerUDP.java from Couleuris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
19 // Reference: EchoServerTCP.java from Couleuris text (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
20
21 public class VerifyingServerTCP {
22     // Map to store user IDs and their corresponding values
23     static Map<String, Integer> UserID_Value_Map = new TreeMap<>();
24     static Socket clientSocket = null; // Declare and Initialize Client Socket
25     static BigInteger e; // e is the exponent of the public key
26     static BigInteger n; // n is the modulus for both the private and public keys
27     static String operation;
28     static String value;
29     static String userId;
```

```
30
31 > public static void main(String args[]) {
32
33     System.out.println("The TCP server is running.");
34
35     try {
36
37         Scanner sc = new Scanner(System.in);
38         System.out.println("Provide port number server should listen on");
39         int serverPort = sc.nextInt(); // Read the port number from the user.
40         ServerSocket listenSocket = new ServerSocket(serverPort); //Initialize Server Socket
41
42         /*
43          * Block waiting for a new connection request from a client.
44          * When the request is received, "accept" it, and the rest
45          * the tcp protocol handshake will then take place, making
46          * the socket ready for reading and writing.
47          */
48         clientSocket = listenSocket.accept();
49         // If we get here, then we are now connected to a client.
50
51         // Set up "in" to read from the client socket
52         Scanner in;
53         in = new Scanner(clientSocket.getInputStream());
54
55         // Set up "out" to write to the client socket
56         PrintWriter out;
57         out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));
58
59         /*
60          * Forever,
61          *   read a line from the socket
62          *   print it to the console
63          *   echo it (i.e. write it) back to the client
64          */
65
66         while (true) {
67             if (in.hasNextLine()) {
68                 String requestString = in.nextLine();
69
70                 if (verify(requestString)) {
71
72                     System.out.println("Signature is verified");
73
74                     System.out.println("User " + userId + " requested " + operation + " operation.");
75
76                     // Initialize integer variable that will store the updated value
77                     // after the operation is performed on value
78                     int n = 0;
79
80                     // If the operation selected is Addition
81                     if (operation.equalsIgnoreCase("add")) {
82                         n = add(userId, Integer.parseInt(value));
83                     }
84
85                     // If the operation selected is Subtraction
86                     else if (operation.equalsIgnoreCase("subtract")) {
87                         n = subtract(userId, Integer.parseInt(value));
88                     }
89
90                     // If the operation selected is Get
91                     else if (operation.equalsIgnoreCase("get")) {
92                         n = get(userId);
93                     }
94
95                     // If the operation selected is Set
96                     else if (operation.equalsIgnoreCase("set")) {
97                         set(userId, value);
98                     }
99
100                    // Print the result
101                    System.out.println("Result: " + n);
102
103                }
104            }
105        }
106    }
107}
```

```

92             // Print to Console
93             System.out.println("Returning " + n + " to client");
94             out.println(n);
95             out.flush();
96         } else {
97             out.println("Error in request");
98             out.flush();
99         }
100     }
101 }
102 }
103 }
104 catch (SocketException e) {
105     // Handle Socket Exception by displaying an error message
106     System.out.println("Socket: " + e.getMessage());
107 } catch (IOException e) {
108     // Handle IO Exception by displaying an error message
109     System.out.println("IO Exception: " + e.getMessage());
110 } catch (NoSuchAlgorithmException ex) {
111     // Handle Runtime Exception by displaying an error message
112     throw new RuntimeException(ex);
113 } finally {
114     try {
115         // Ensure that the clientSocket is closed, if it's not null
116         if (clientSocket != null) {
117             clientSocket.close();
118         }
119     } catch (IOException e) {
120         // Handle any potential IOException that may occur when closing the socket
121         // This is typically ignored, as the main goal is to ensure the socket is closed
122     }
123 }
124 }
125 }
126 // Adds value received to the specific user
127 public static int add(String userId, int n) {
128     int result;
129     if (UserID_Value_Map.containsKey(userId)) {
130         result = UserID_Value_Map.get(userId) + n;
131         UserID_Value_Map.put(userId, result);
132     }
133     else {
134         result = n;
135         UserID_Value_Map.put(userId, n);
136     }
137     return result;
138 }
139 }
```

```

140 // Subtracts value received from the specific user
141 1usage
142 public static int subtract(String userId, int n) {
143     int result;
144     if (UserID_Value_Map.containsKey(userId)) {
145         result = UserID_Value_Map.get(userId) - n;
146         UserID_Value_Map.put(userId, result);
147     }
148     else {
149         result = n;
150         UserID_Value_Map.put(userId, n);
151     }
152     return result;
153 }
154
155 // Gets value received from the specific user
156 1usage
157 public static int get(String userId) {
158     int result;
159     if (UserID_Value_Map.containsKey(userId)) {
160         result = UserID_Value_Map.get(userId);
161     }
162     else {
163         result = 0;
164         UserID_Value_Map.put(userId, 0);
165     }
166     return result;
167 }
168
169 // Method to verify if encryption was done correctly @throws NoSuchAlgorithmException
170 // @throws UnsupportedEncodingException
171 1usage
172 public static boolean verify(String message) throws NoSuchAlgorithmException, UnsupportedEncodingException {
173     boolean verify = true;
174
175     // Fetch different values from the string
176     String id = message.split( regex: ",") [0];
177     String publicKey = message.split( regex: ",") [1];
178     operation = message.split( regex: ",") [2];
179     value = message.split( regex: ",") [3];
180     String sign = message.split( regex: ",") [4];
181
182     // Fetch the public and modulus , public key is first 6 characters
183     e = new BigInteger(publicKey.substring(0, 5));
184     n = new BigInteger(publicKey.substring( beginIndex: 5));
185
186     System.out.println("public key received from client " + e+n);
187
188     // Validate id
189     String validId = validateId(publicKey);
190
191     if (!validId.equals(id))
192         verify = false;
193
194     // Recreate message
195     String messageToCheck = id + "," + String.valueOf(e) + String.valueOf(n) + "," + operation + "," + value;

```

```

195         // Verify if signature is valid
196         if (!sign(messageToCheck, sign))
197             verify = false;
198
199         if (verify)
200             userId = validId;
201
202     return verify;
203 }
204
205
206 // Method to convert bytes to hex
207 // Reference Code: https://mkyong.com/java/java-how-to-convert-bytes-to-hex/
208 @
209     public static String bytesToHex(byte[] bytes) {
210
211         StringBuilder result = new StringBuilder();
212
213         // Step 1: Iterate through each byte in the 'bytes' array.
214         for (byte aByte : bytes) {
215
216             // Step 2: Convert the byte to an integer and apply a bitwise AND operation with 0xff.
217             int decimal = (int) aByte & 0xff; // This ensures that the integer is non-negative.
218
219             // Step 3: Convert the decimal value to a hexadecimal string.
220             String hex = Integer.toHexString(decimal);
221
222             // Step 4: Check if the hex string has an odd length, and if so, pad it with a leading '0'.
223             if (hex.length() % 2 == 1) {
224                 hex = "0" + hex;
225             }
226
227             // Step 5: Append the hexadecimal value to the result.
228             result.append(hex);
229
230         }
231
232         // Step 6: Return the concatenated hexadecimal string representation of the bytes.
233         return result.toString();
234     }
235
236
237 // Method to validate the user id
238 @
239     public static String validateId(String key) throws NoSuchAlgorithmException {
240
241         // Step 1: Create a MessageDigest instance using SHA-256 hashing algorithm.
242         MessageDigest digest = MessageDigest.getInstance(algorithm: "SHA-256");
243
244         // Step 2: Generate a hash of the input 'key'.
245         byte[] hashBytes = digest.digest(key.getBytes());
246
247         // Step 3: Create a new byte array to store the final 20 bytes of the hash.
248         byte[] finalBytes = new byte[20];
249
250         // Step 4: Copy the least significant 20 bytes of the hash into the final array.
251         // This is done to ensure that the output is a 20-byte (160-bit) hash.
252         System.arraycopy(hashBytes, srcPos: hashBytes.length - 20, finalBytes, destPos: 0, length: 20);
253
254         // Step 5: Convert the byte array to a hexadecimal string representation.
255         return bytesToHex(finalBytes);
256     }

```

```
251
252     // Method to check whether signed message is valid
253     // Reference Code : ShortMessageVerify.java (https://github.com/CMU-Heinz-95702/Project-2-Client-Server)
254     @
255         public static boolean sign(String messageToCheck, String encryptedHashStr) throws
256             UnsupportedEncodingException, NoSuchAlgorithmException {
257
258             // Take the encrypted string and make it a big integer
259             BigInteger encryptedHash = new BigInteger(encryptedHashStr);
260
261             // Decrypt the encryptedHash to compute a decryptedHash
262             BigInteger decryptedHash = encryptedHash.modPow(e, n);
263
264             // Get the bytes from messageToCheck
265             byte[] bytesOfMessageToCheck = messageToCheck.getBytes( charsetName: "UTF-8");
266
267             // Hash the messageToCheck using SHA-256 (be sure to handle the extra byte as described in the signing method.)
268             MessageDigest md = MessageDigest.getInstance( algorithm: "SHA-256");
269             byte[] messageToCheckDigest = md.digest(bytesOfMessageToCheck);
270             messageToCheckDigest[0] = 0;
271
272             // Make it a big int
273             BigInteger bigIntegerToCheck = new BigInteger(messageToCheckDigest);
274
275             // If this new hash is equal to the decryptedHash, return true else false.
276             if (bigIntegerToCheck.compareTo(decryptedHash) == 0) {
277                 return true;
278             }
279             else {
280                 return false;
281             }
282         }
283     }
```

Project2Task5ClientConsole

```
Run VerifyingServerTCP × SigningClientTCP ×
G | D | E | C | : 
/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/Lib/idea_rt.jar=52371:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/ariamecorrea/IdeaProjects/Project2Task5/target/classes com.example.project2task5.SigningClientTCP
The TCP client is running.
Provide server side port number
6789
Public key =
65537224359692681283326195129604804066071088143237067408219964348454299704607447426790894637773107245231580225316702099242183489416787855474597360419524730653294531406
772694755274776233961423631788150447862153416915965088015678739443293863862203
Private key =
180817404642263495780617902353497440266315966483457701177608933262154171783239822412880935959511095131060939279666108818369052620927609208049905133875714078558847195608
328299439363269422812403602408473574730733203667522660667095260846646907322435969268128332619512960480406607108814323706740821996434845429970460744742679089463777310724
5231580225316702099242183489416787855474597360419524730653294531486772694755274776233961423631788150447862153416915965088015678739443293863862203
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:
10
Received: 10
The server returned 10
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:
5
Received: 5
The server returned 5
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Received: 5
The server returned 5
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
```

Project2Task5ServerConsole

```
Run VerifyingServerTCP × SigningClientTCP ×

/Library/Java/JavaVirtualMachines/temurin-17.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA.app/Contents/lib/idea_rt.jar=52368:/Applications/IntelliJ IDEA.app/Contents/bin -Dfile.encoding=UTF-8 -classpath /Users/ariane.correa/IdeaProjects/Project2Task5/target/classes com.example.project2task5.VerifyingServerTCP
The TCP server is running.
Provide port number server should listen on
6789
public key received from client
655372243596926812833261951296048040660710881432370674082199643484542997046074474267908946377731072452315802253167020992421834489416787855474597360419524730653294531406
772694755274776233961423631788150447862153416915965088015678739443293063862203
Signature is verified
User a13778b13cb7a49d9e69bbeb71df079db016a5cb requested add operation.
Returning 10 to client
public key received from client
655372243596926812833261951296048040660710881432370674082199643484542997046074474267908946377731072452315802253167020992421834489416787855474597360419524730653294531406
772694755274776233961423631788150447862153416915965088015678739443293063862203
Signature is verified
User a13778b13cb7a49d9e69bbeb71df079db016a5cb requested subtract operation.
Returning 5 to client
public key received from client
655372243596926812833261951296048040660710881432370674082199643484542997046074474267908946377731072452315802253167020992421834489416787855474597360419524730653294531406
772694755274776233961423631788150447862153416915965088015678739443293063862203
Signature is verified
User a13778b13cb7a49d9e69bbeb71df079db016a5cb requested get operation.
Returning 5 to client
|
```