Author: Shivam Patel
Andrew ID: shpatel
Email Address: shpatel@cmu.edu
Last Modified: October 7, 2022
Project 2

Project 2 – Task 0

Project2Task0Client

```java
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Last Modified: October 7, 2022
 * File: EchoClientUDP.java
 * Part Of: Project2Task0
 * Partial code modified from: https://github.com/CMU-Heinz-95702/Project-2-Client-Server
 *
 * This Java file acts as a UDP Client which simply sends a
 * message to the server and print the message from the server
 * to the user. The server in this case is just echoing back
 * client's request to the client. So, the client will output
 * whatever it sent to the server. It sends a packet to the
 * server and waits for the server to execute the requested
 * operation. When the response packet arrives from the server,
 * the client creates a String object and displays the output
 * to the user. If the client sends a message of "halt!", to
 * the server, the server will halt its execution and relay
 * back the same message to the client. Hence, the client will
 * also halt its execution.
 */

// imports required for UDP/IP and IO operations
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class EchoClientUDP{
    /***
     * The main method for the client to request for the server port from the user,
     * request a message from the user, send a request message to the server, receive
     * a reply from the server and print the reply to the user
     * @param args Command line arguments (none here)
     */
    public static void main(String args[]){

        // Prompting the user that the client is running
        System.out.println("The client is running.");

        // Creating a Scanner object for taking inputs from the user
        Scanner s = new Scanner(System.in);

        // Requesting the user for the server side port
        System.out.print("Enter server side port number: ");

        // Getting the server port from the user
```

```
        int serverPort = s.nextInt();

        // Define a UDP style Datagram socket
        DatagramSocket aSocket = null;
        try {
            // Creating a String object for localhost
            String localhost = "";

            // Build an InetAddress object from a DNS name
            InetAddress aHost = InetAddress.getByName(localhost);

            // Creating a new DatagramSocket
            aSocket = new DatagramSocket();

            // Stores the message from the user
            String nextLine;

            // Prompt the user to enter a message to send to the server
            System.out.print("Enter request message to server: ");

            // Create a BufferReader object to get input from the user
            BufferedReader typed = new BufferedReader(new
InputStreamReader(System.in));

            // Until the user input is not null, read an input message from
the user
            while ((nextLine = typed.readLine()) != null) {

                // Create a byte array of the user input
                byte [] m = nextLine.getBytes();

                // Build the packet holding the byte array of user input,
its length,
                // destination address, and port
                DatagramPacket request = new DatagramPacket(m,  m.length,
aHost, serverPort);

                // Send the Datagram on the socket to the server
                aSocket.send(request);

                // Build a byte array buffer of the maximum size possible
for the reply
                // The maximum size is set by aSocket.getSendBufferSize()
                byte[] buffer = new byte[aSocket.getSendBufferSize()];

                // Build a DatagramPacket for the reply
                DatagramPacket reply = new DatagramPacket(buffer,
buffer.length);

                // Block and wait to receive the reply
                aSocket.receive(reply);

                // Convert the data of the reply into byte array
                byte[] reply_data_bytes = reply.getData();

                // Create a new byte array of only the required length
(necessary for the message)
                byte[] new_reply_data_bytes = new byte[reply.getLength()];

                // Copy content from data_bytes to new_data_bytes until the
required length
```

```
                System.arraycopy(reply_data_bytes, 0, new_reply_data_bytes,
0, reply.getLength());

                // Convert the reply byte array into String
                String replyString = new String(new_reply_data_bytes);

                // Print the reply to the user
                System.out.println("Reply: " + replyString);

                // If the reply from the server is not "halt!"
                if (!replyString.equals("halt!")) {
                    // Prompt the user for another message
                    System.out.print("Enter request message to server: ");
                }
                // If the server request the client to halt
                else {
                    // Prompt the user that the client is quitting
                    System.out.println("Client side quitting");
                    // Halt client execution
                    System.exit(0);
                }
            }

        }
        // Handle socket exceptions
        catch (SocketException e) {System.out.println("Socket: " +
e.getMessage());
        }
        // Handle general I/O exceptions
        catch (IOException e){System.out.println("IO: " + e.getMessage());
        }
        // Always close the socket
        finally {if(aSocket != null) aSocket.close();}
    }
}
```

Project2Task0Server

```java
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Last Modified: October 7, 2022
 * File: EchoServerUDP.java
 * Part Of: Project2Task0
 * Partial code modified from: https://github.com/CMU-Heinz-95702/Project-2-Client-Server
 *
 * This Java file acts as a UDP Server which simply echoes
 * back to the client the request it received (message) from
 * the client. It starts off by requesting the user for the
 * port number it should be listening to. Then it creates a
 * DatagramSocket and receives a request from the client in
 * the form of a DatagramPacket. The message from the client
 * is in the form of a byte array which is converted to String
 * and displayed to the user. The same request message is then
 * sent to the client. However, if the client sends a message
 * of "halt!", the server will halt its execution.
 */

// Imports required for UDP/IP and IO operations
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class EchoServerUDP{
    /***
     * The main method for the server to set its port, connect
     * with the client, receive a request from the client and
     * perform its echo operation to the client
     * @param args Command line arguments (none here)
     */
    public static void main(String args[]){

        // Prompting the user that the server is running
        System.out.println("The server is running.");

        // Creating a Scanner object for taking inputs from the user
        Scanner s = new Scanner(System.in);

        // Requesting the user for the port that the server should listen to
        System.out.print("Enter port number that the server is supposed to listen to: ");

        // Getting the server port from the user
        int port = s.nextInt();

        // Define a UDP style Datagram socket and set the server port to it
        try (DatagramSocket aSocket = new DatagramSocket(port)) {

            // Keep listening to the client's request
            while (true) {

                // Build a byte array buffer of the maximum size possible for the request
                // The maximum size is set by aSocket.getReceiveBufferSize()
```

```
                byte[] buffer = new byte[aSocket.getReceiveBufferSize()];

                // Build a DatagramPacket for the request
                DatagramPacket request = new DatagramPacket(buffer,
buffer.length);

                // Block and wait to receive the client request at the port
                aSocket.receive(request);

                // Build a DatagramPacket for the reply containing the
request data from the client
                // The packet includes data in byte array, length of data,
destination address and port
                DatagramPacket reply = new
DatagramPacket(request.getData(),
                        request.getLength(), request.getAddress(),
request.getPort());

                // Get the byte array from client's request
                byte[] data_bytes = request.getData();

                // Create a new byte array of only the required length
(necessary for the message)
                byte[] new_data_bytes = new byte[request.getLength()];

                // Copy content from data_bytes to new_data_bytes until the
required length
                System.arraycopy(data_bytes, 0, new_data_bytes, 0,
request.getLength());

                // Convert the message from the client into String form
                String requestString = new String(new_data_bytes);

                // Print (echo) the message of the client to the user
                System.out.println("Echoing: " + requestString);

                // Send a reply to the client (with the same contents as
the request)
                aSocket.send(reply);

                // If client messaged to halt
                if (requestString.equals("halt!")) {
                    // Prompt the user that the server is quitting
                    System.out.println("Server side quitting");
                    // Halt server execution
                    System.exit(0);
                }
            }
        }
        // Handle socket exceptions
        catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        }
        // Handle general I/O exceptions
        catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        }
    }
}
```

Project2Task0ClientConsole

```
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute
d Systems for ISM\Projects\Project2\Project2Task0\src> java .\EchoClientUDP
.java
The client is running.
Enter server side port number: 6789
Enter request message to server: line 1
Reply: line 1
Enter request message to server: line 2
Reply: line 2
Enter request message to server: line 3
Reply: line 3
Enter request message to server: line 4
Reply: line 4
Enter request message to server: line 5
Reply: line 5
Enter request message to server: halt!
Reply: halt!
Client side quitting
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute
d Systems for ISM\Projects\Project2\Project2Task0\src>
```

Project2Task0ServerConsole

```
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distributed Systems for ISM\Projects\Project2\Project2Task0\src> java .\EchoServerUDP.java
The server is running.
Enter port number that the server is supposed to listen to: 6789
Echoing: line 1
Echoing: line 2
Echoing: line 3
Echoing: line 4
Echoing: line 5
Echoing: halt!
Server side quitting
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distributed Systems for ISM\Projects\Project2\Project2Task0\src>
```

Project 2 – Task 1

Project2Task1Eavesdropper

```java
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Last Modified: October 7, 2022
 * File: EavesdropperUDP.java
 * Part Of: Project2Task1
 * Partial code modified from: https://github.com/CMU-Heinz-95702/Project-2-Client-Server
 *
 * This Java file acts as a UDP Eavesdropper which acts as a passive
 * Eavesdropper between the UDP Client and the UDP Server. After
 * running, the Eavesdropper will ask the user for two ports. One port
 * will be the port that the EavesdropperUDP.java will listen on and the
 * other port will be the port number of the server that Eavesdropper.java
 * is masquerading as. The Eavesdropper will display all the messages that
 * go through it (both from the Server and the Client). However, if the
 * client requested for a "halt!" request, it will display a line of
 * asterisks, pass the message to the EchoServer, receive the echo
 * from the Server, and pass the echo to the Client. In the halt execution,
 * both the client and the server will halt their executions, but the
 * Eavesdropper will run forever. Moreover, if the client directly connects
 * to the server, the Eavesdropper will not come into the picture.
 */

// imports required for UDP/IP and IO operations
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class EavesdropperUDP {

    /***
     * The main method for the Eavesdropper to act as a passive malicious player in the middle.
     * It sets the port number of the server and itself, receives the request from the client,
     * passses the request to the server, receive the reply from the server and passes the reply
     * for the client.
     * @param args Command line arguments (none here)
     */
    public static void main(String args[]){

        // Prompting the user that the Eavesdropper is running
        System.out.println("The Eavesdropper is running.");

        // Creating a Scanner object for taking inputs from the user
        Scanner s = new Scanner(System.in);

        // Requesting the user for Eavesdropper port
        System.out.print("Enter evasdropper port number: ");

        // Getting the Eavesdropper port from the user
        int eavesdropperPort = s.nextInt();

        // Requesting the user for the server side port
```

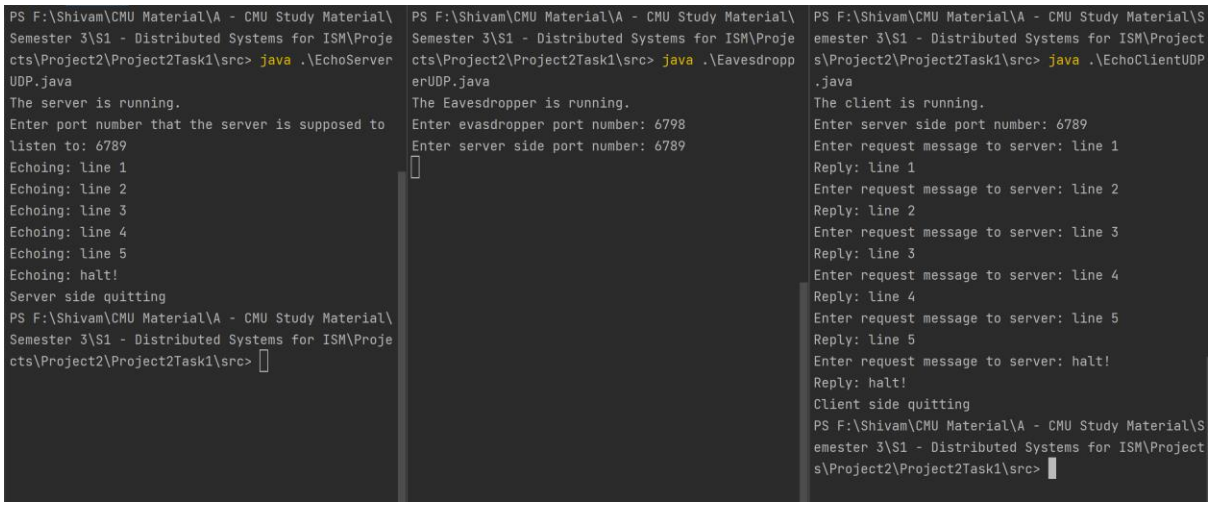
```
        System.out.print("Enter server side port number: ");

        // Getting the server port from the user
        int serverPort = s.nextInt();

        // Define a UDP style Datagram socket and set the server port to it
        try (DatagramSocket eavesdropper_Socket = new
DatagramSocket(eavesdropperPort)) {

            // Keep listening to the client's request
            while (true) {

                // Part 1: Request from client
                // Build a byte array buffer of the maximum size possible
for the request
                // The maximum size is set by
aSocket.getReceiveBufferSize()
                byte[] buffer = new
byte[eavesdropper_Socket.getReceiveBufferSize()];

                // Build a DatagramPacket for the request from client
                DatagramPacket evasdropper_request_from_client = new
DatagramPacket(buffer, buffer.length);

                // Block and wait to receive the client request at the port

eavesdropper_Socket.receive(evasdropper_request_from_client);

                // Build a DatagramPacket for the request to the server
containing the request data
                // from the client. The packet includes data in byte array,
length of data, destination
                // address and port
                DatagramPacket request_to_server = new
DatagramPacket(evasdropper_request_from_client.getData(),
                        evasdropper_request_from_client.getLength(),
evasdropper_request_from_client.getAddress(),
                        serverPort);

                // Get the byte array from client's request
                byte[] request_data_bytes_from_client =
evasdropper_request_from_client.getData();

                // Create a new byte array of only the required length
(necessary for the message)
                byte[] new_request_data_bytes_from_client = new
byte[evasdropper_request_from_client.getLength()];

                // Copy content from data_bytes to new_data_bytes until the
required length
                System.arraycopy(request_data_bytes_from_client, 0,
new_request_data_bytes_from_client, 0,
evasdropper_request_from_client.getLength());

                // Convert the message from the client into String form
                String evasdropper_requestString_from_client = new
String(new_request_data_bytes_from_client);

                // If the client requested for a "halt!"
                if (evasdropper_requestString_from_client.equals("halt!"))
{
```

```
                    // Print a special message to the user (a line of
asterisks)
                    System.out.println("******************************");
                }

                // Print (echo) the message of the client to the user
                System.out.println("Request message from client = " +
evasdropper_requestString_from_client);

                // Part 2: Request to server
                // Send a request to the server (with the same contents as
the request)
                eavesdropper_Socket.send(request_to_server);

                // Part 3: Reply from server
                // Build a byte array buffer of the maximum size possible
for the reply
                // The maximum size is set by aSocket.getSendBufferSize()
                buffer = new byte[eavesdropper_Socket.getSendBufferSize()];

                // Build a DatagramPacket for the reply
                DatagramPacket reply_from_server = new
DatagramPacket(buffer, buffer.length);

                // Block and wait to receive the reply
                eavesdropper_Socket.receive(reply_from_server);

                // Convert the data of the reply into byte array
                byte[] reply_data_bytes_from_server =
reply_from_server.getData();

                // Create a new byte array of only the required length
(necessary for the message)
                byte[] new_reply_data_bytes_from_server = new
byte[reply_from_server.getLength()];

                // Copy content from data_bytes to new_data_bytes until the
required length
                System.arraycopy(reply_data_bytes_from_server, 0,
new_reply_data_bytes_from_server, 0, reply_from_server.getLength());

                // Convert the reply byte array into String
                String evasdropper_replyString_from_server = new
String(new_reply_data_bytes_from_server);

                // Print the reply to the user
                System.out.println("Reply message from server: " +
evasdropper_replyString_from_server);
                System.out.println();

                // Part 4: Reply to client
                // Build a DatagramPacket for the reply containing the
request data from the client
                // The packet includes data in byte array, length of data,
destination address and port
                DatagramPacket reply_to_client = new
DatagramPacket(reply_from_server.getData(),
                        reply_from_server.getLength(),
reply_from_server.getAddress(), evasdropper_request_from_client.getPort());

                // Send a reply to the client (with the contents from the
```

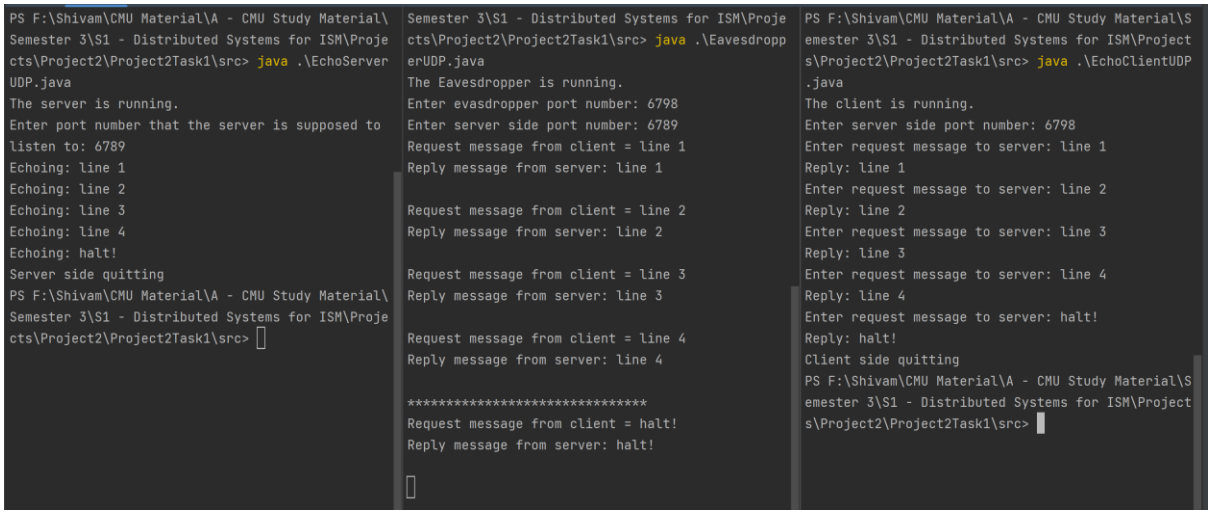
```
server)
                eavesdropper_Socket.send(reply_to_client);
            }
        }
        // Handle socket exceptions
        catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        }
        // Handle general I/O exceptions
        catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        }
    }
}
```

Project2Task1ThreeConsoles

Client using port 6789 (correct port)



Client using port 6798 (malicious port)



Project 2 – Task 2

Project2Task2Client

```java
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Last Modified: October 7, 2022
 * File: AddingClientUDP.java
 * Part Of: Project2Task2
 * Partial code modified from: https://github.com/CMU-Heinz-95702/Project-2-Client-Server
 *
 * This Java file acts as a UDP Client which sends a value to
 * the server. The value is added to a sum variable in the sum
 * and the updated sum is returned to the client. The client
 * receives the value of the updated sum and prints it to the
 * user. The client sends a packet to the server and waits for
 * the server to execute the requested operation. When the response
 * packet arrives from the server, the client creates an int
 * object and displays the output to the user. If the client
 * sends a message of "halt!" to the server, the server will not
 * be affected; however, the client will stop its execution.
 */

// imports required for UDP/IP and IO operations
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class AddingClientUDP {

    // Stores the value of port number of the server
    static int serverPort;

    /***
     * The main method for the client to request for the server port from the user,
     * request a message from the user, send a request message to the server, receive
     * a reply from the server and print the reply to the user
     * @param args Command line arguments (none here)
     */
    public static void main(String args[]) throws IOException {

        // Prompting the user that the client is running
        System.out.println("The client is running.");

        // Creating a Scanner object for taking inputs from the user
        Scanner s = new Scanner(System.in);

        // Requesting the user for the server side port
        System.out.print("Please enter server port: ");

        // Getting the server port from the user
        serverPort = s.nextInt();
        System.out.println();

        // Stores the input from the user
        String nextLine;
```

```
        // Create a BufferReader object to get input from the user
        BufferedReader typed = new BufferedReader(new
InputStreamReader(System.in));

        // Until the user input is not null, read an input message from the
user
        while ((nextLine = typed.readLine()) != null) {

            // Create a byte array of the user input
            byte[] m = nextLine.getBytes();

            // If client request for a "halt!"
            if (nextLine.equals("halt!")) {
                // Prompt the user that the client is quitting
                System.out.println("Client side quitting.\n");
                // Halt client execution
                System.exit(0);
            }

            // Convert byte array of input to String
            String user_string = new String(m);

            // Parse user String to int
            int user_int = Integer.parseInt(user_string);

            // Request the addition operation from server and store the
value of sum
            int serverSumReturned = add(user_int);

            // Display the sum received from the server to the user
            System.out.println("The server returned " + serverSumReturned +
".");
        }
    }

    /***
     * Function to communicate with the server and perform the addition
     * operation on the requested integer value by the client
     * @param i Integer value to add to the sum
     * @return Updated sum from the server
     */
    public static int add(int i) {

        // Define a UDP style Datagram socket
        DatagramSocket aSocket = null;

        // Stores the sum returned from the server
        int serverSumReturned = 0;
        try {
            // Creating a String object for localhost
            String localhost = "";

            // Build an InetAddress object from a DNS name
            InetAddress aHost = InetAddress.getByName(localhost);

            // Creating a new DatagramSocket
            aSocket = new DatagramSocket();

            // Convert user int to byte array
            // Source: https://www.baeldung.com/java-byte-array-to-
```

```
number#:~:text=The%20Ints%20class%20also%20has,toByteArray(value)%3B
            byte[] user_int_bytes = new byte[Integer.BYTES];
            int length = user_int_bytes.length;
            for (int j = 0; j < length; j++) {
                user_int_bytes[length - j - 1] = (byte) (i & 0xFF);
                i >>= 8;
            }

            // Build the packet holding the byte array of user input, its
length,
            // destination address, and port
            DatagramPacket request = new DatagramPacket(user_int_bytes,
user_int_bytes.length,
                    aHost, serverPort);

            // Send the Datagram on the socket to the server
            aSocket.send(request);

            // Build a byte array buffer of the maximum size possible for
the reply
            // The maximum size is set by aSocket.getSendBufferSize()
            byte[] buffer = new byte[aSocket.getSendBufferSize()];

            // Build a DatagramPacket for the reply
            DatagramPacket reply = new DatagramPacket(buffer,
buffer.length);

            // Block and wait to receive the reply
            aSocket.receive(reply);

            // Convert the data of the reply into byte array
            byte[] reply_data_bytes = reply.getData();

            // Create a new byte array of only the required length
(necessary for the message)
            byte[] new_reply_data_bytes = new byte[reply.getLength()];

            // Copy content from data_bytes to new_data_bytes until the
required length
            System.arraycopy(reply_data_bytes, 0, new_reply_data_bytes, 0,
reply.getLength());

            // Convert the byte array of the reply into int
            // Source: https://www.baeldung.com/java-byte-array-to-
number#:~:text=The%20Ints%20class%20also%20has,toByteArray(value)%3B
            for (byte b : new_reply_data_bytes) {
                serverSumReturned = (serverSumReturned << 8) + (b & 0xFF);
            }

        }
        // Handle socket exceptions
        catch (
                SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        }
        // Handle general I/O exceptions
        catch (
                IOException e) {
            System.out.println("IO: " + e.getMessage());
        }
        // Always close the socket
```

```
        finally {
            if (aSocket != null) aSocket.close();
        }

        // Return the updated sum
        return serverSumReturned;
    }
}
```

Project2Task2Server

```
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Last Modified: October 7, 2022
 * File: AddingServerUDP.java
 * Part Of: Project2Task2
 * Partial code modified from: https://github.com/CMU-Heinz-95702/Project-
2-Client-Server
 *
 * This Java file acts as a UDP Server which add the values sent by
 * the client and stores them to a variable 'sum'. After performing
 * addition for each request from the client, it echoes back the sum
 * to the client. The port number that it would listen to is preset.
 * It creates a DatagramSocket and receives a request from the client
 * in the form of a DatagramPacket. The message from the client is in
 * the form of a byte array which is converted to int for performing
 * the operation and the sum is displayed to the user. The sum is
 * then sent to the client. However, if the client sends a message
 * of "halt!", the server will not halt its execution and will keep the
 * value of the 'sum' variable as it was before the client requested
 * for the halt operation. After the client resumes, the server will
 * keep updating the value of the previous sum.
 */

// Imports required for UDP/IP and IO operations
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class AddingServerUDP{

    // Stores the value of the sum
    static int sum = 0;

    /***
     * The main method for the server to set its port, connect
     * with the client, receive a request from the client, perform
     * the addition operation and reply the sum to the client
     * @param args Command line arguments (none here)
     */
    public static void main(String args[]){

        // Prompting the user that the server is running
        System.out.println("Server started");

        // Hard coded port for the server (as suggest on Piazza)
        int port = 6789;

        // Define a UDP style Datagram socket and set the server port to it
        try (DatagramSocket aSocket = new DatagramSocket(port)) {

            // Keep listening to the client's request
            while (true) {

                // Build a byte array buffer of the maximum size possible
for the request
                // The maximum size is set by
aSocket.getReceiveBufferSize()
                byte[] buffer = new byte[aSocket.getReceiveBufferSize()];
```

```
                // Build a DatagramPacket for the request
                DatagramPacket request = new DatagramPacket(buffer,
buffer.length);

                // Block and wait to receive the client request at the port
                aSocket.receive(request);

                // Get the byte array from client's request
                byte[] request_data_bytes = request.getData();

                // Create a new byte array of only the required length
(necessary for the message)
                byte[] new_request_data_bytes = new
byte[request.getLength()];

                // Copy content from data_bytes to new_data_bytes until the
required length
                System.arraycopy(request_data_bytes, 0,
new_request_data_bytes, 0, request.getLength());

                // Store the value sent by the client
                int requestInt = 0;

                // Convert byte array to integer
                // Source: https://www.baeldung.com/java-byte-array-to-
number#:~:text=The%20Ints%20class%20also%20has,toByteArray(value)%3B
                for (byte b : new_request_data_bytes) {
                    requestInt = (requestInt << 8) + (b & 0xFF);
                }

                // Perform the addition operation and the value into the
'sum' variable
                sum = serverAdd(requestInt, sum);

                // Display the sum to the user and state that the sum is
being returned to the client
                System.out.println("Returning sum of " + sum + " to
client\n");

                // Convert int sum into byte array
                // Source: https://www.baeldung.com/java-byte-array-to-
number#:~:text=The%20Ints%20class%20also%20has,toByteArray(value)%3B
                int sum_copy = sum; // Create a copy if the sum and convert
the copy into byte array
                byte[] bytes_reply = new byte[Integer.BYTES];
                int length = bytes_reply.length;
                for (int i = 0; i < length; i++) {
                    bytes_reply[length - i - 1] = (byte) (sum_copy & 0xFF);
                    sum_copy >>= 8;
                }

                // Build a DatagramPacket for the reply containing the sum
data after the addition operation
                // The packet includes data in byte array, length of data,
destination address and port
                DatagramPacket reply = new DatagramPacket(bytes_reply,
                        bytes_reply.length, request.getAddress(),
request.getPort());

                // Send a reply to the client (with the value of the
```

```
updated sum)
                aSocket.send(reply);
            }
        }
        // Handle socket exceptions
        catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        }
        // Handle general I/O exceptions
        catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        }
    }

    /***
     * Function to perform the addition operation requested by the client
     * @param i Stores the value to add to the client
     * @param sum Stores the current value of the sum variable
     * @return Updated value of the sum variable
     */
    public static int serverAdd(int i, int sum) {

        // Prompt the user about the addition operation
        System.out.println("Adding: " + i + " to " + sum);

        // Return the updated sum
        return i + sum;
    }
}
```

Project2Task2ClientConsole

```
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute
d Systems for ISM\Projects\Project2\Project2Task2\src> java .\AddingClientU
DP.java
The client is running.
Please enter server port: 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!
Client side quitting.

PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute
d Systems for ISM\Projects\Project2\Project2Task2\src> java .\AddingClientU
DP.java
The client is running.
```

(client console continuation)

```
The server returned 9.
halt!
Client side quitting.

PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute
d Systems for ISM\Projects\Project2\Project2Task2\src> java .\AddingClientU
DP.java
The client is running.
Please enter server port: 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!
Client side quitting.
```

Project2Task2ServerConsole

```
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distributed Systems for ISM\Projects\Project2\Project2Task2\src> java .\AddingServerUDP.java
Server started
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 – Task 3

Project2Task3Client

```java
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Last Modified: October 7, 2022
 * File: RemoteVariableClientUDP.java
 * Part Of: Project2Task3
 * Partial code modified from: https://github.com/CMU-Heinz-95702/Project-2-Client-Server
 *
 * This Java file acts as a UDP Client which sends a request to
 * the server. The request is made of client ID, operation (add,
 * subtract, get) and operand. The operand is added/subtracted
 * from the sum and the updated sum is returned to the client or
 * the sum is directly returned from the server if the client made
 * a get request. The client receives the value of the sum (of the
 * client ID) and prints it to the user. The client sends a packet
 * to the server and waits for the server to execute the requested
 * operation. When the response packet arrives from the server, the
 * client creates an int object and displays the output to the user.
 * If the client wishes to halt its execution, the server will not
 * be affected.
 */

// imports required for UDP/IP and IO operations
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class RemoteVariableClientUDP {

    // Stores the value of port number of the server
    static int serverPort;

    /***
     * The main method for the client to request for the server port from the user,
     * request a message (client ID, operation, operand) from the user, send a request
     * message to the server, receive a reply from the server (sum of the client ID)
     * and print the reply to the user
     * @param args Command line arguments (none here)
     */
    public static void main(String args[]) throws IOException {

        // Prompting the user that the client is running
        System.out.println("\nThe client is running.");

        // Creating a Scanner object for taking inputs from the user
        Scanner s = new Scanner(System.in);

        // Requesting the user for the server side port
        System.out.print("Please enter server port: ");

        // Getting the server port from the user
        serverPort = s.nextInt();
```

```
        System.out.println();

        // Stores the input from the user
        String user_input = "";

        // Create a BufferReader object to get input from the user
        BufferedReader typed = new BufferedReader(new
InputStreamReader(System.in));

        // Keeping executing until the client is terminated
        while (true) {

            // Initialize user input to null
            user_input = "";

            // Prompt the user for operation
            System.out.println("""
                    1. Add a value to your sum.
                    2. Subtract a value from your sum.
                    3. Get your sum.
                    4. Exit client""");

            // Update user input
            user_input = user_input + typed.readLine();

            // Switch case for user input
            switch (user_input) {

                // If user requested for an addition operation
                case "1" -> {

                    // Prompt the user for an operand
                    System.out.println("Enter value to add:");

                    // Update user input
                    user_input = user_input + "," + typed.readLine() + ",";
                }

                // If user requested for a subtraction operation
                case "2" -> {
                    // Prompt the user for an operand
                    System.out.println("Enter value to subtract:");

                    // Update user input
                    user_input = user_input + "," + typed.readLine() + ",";
                }

                // If user requested for a get operation
                case "3" -> user_input = user_input + ",,";

                // If user requested for halting
                case "4" -> {
                    // Prompt the user that the client is quitting
                    System.out.println("Client side quitting. The remote
variable server is still running.");
                    // Halt client execution
                    System.exit(0);
                }
            }

            // Prompt user for client ID
```

```
            System.out.println("Enter your ID:");

            // Update user input
            user_input = user_input + typed.readLine();

            // Request the operation from server and store the value of sum
            int serverSumReturned = operations(user_input);

            // Display the sum received from the server to the user
            System.out.println("The result is " + serverSumReturned +
".\n");
        }
    }

    /***
     * Function to communicate with the server and perform the required
     * operation on the requested integer value by the client
     * @param user_input Input from the user containing client ID,
operation and operand
     * @return Updated sum from the server
     */
    public static int operations(String user_input) {

        // Convert user_input to byte array
        byte[] user_inputs_in_bytes = user_input.getBytes();

        // Define a UDP style Datagram socket
        DatagramSocket aSocket = null;

        // Stores the sum returned from the server
        int serverSumReturned = 0;
        try {
            // Creating a String object for localhost
            String localhost = "";

            // Build an InetAddress object from a DNS name
            InetAddress aHost = InetAddress.getByName(localhost);

            // Creating a new DatagramSocket
            aSocket = new DatagramSocket();

            // Build the packet holding the byte array of user input, its
length,
            // destination address, and port
            DatagramPacket request = new
DatagramPacket(user_inputs_in_bytes, user_inputs_in_bytes.length,
                    aHost, serverPort);

            // Send the Datagram on the socket to the server
            aSocket.send(request);

            // Build a byte array buffer of the maximum size possible for
the reply
            // The maximum size is set by aSocket.getSendBufferSize()
            byte[] buffer = new byte[aSocket.getSendBufferSize()];

            // Build a DatagramPacket for the reply
            DatagramPacket reply = new DatagramPacket(buffer,
buffer.length);

            // Block and wait to receive the reply
```

```
            aSocket.receive(reply);

            // Convert the data of the reply into byte array
            byte[] reply_data_bytes = reply.getData();

            // Create a new byte array of only the required length
(necessary for the message)
            byte[] new_reply_data_bytes = new byte[reply.getLength()];

            // Copy content from data_bytes to new_data_bytes until the
required length
            System.arraycopy(reply_data_bytes, 0, new_reply_data_bytes, 0,
reply.getLength());

            // Convert the byte array of the reply into int
            // Source: https://www.baeldung.com/java-byte-array-to-
number#:~:text=The%20Ints%20class%20also%20has,toByteArray(value)%3B
            for (byte b : new_reply_data_bytes) {
                serverSumReturned = (serverSumReturned << 8) + (b & 0xFF);
            }

        }
        // Handle socket exceptions
        catch (
                SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        }
        // Handle general I/O exceptions
        catch (
                IOException e) {
            System.out.println("IO: " + e.getMessage());
        }
        // Always close the socket
        finally {
            if (aSocket != null) aSocket.close();
        }

        // Return the updated sum
        return serverSumReturned;
    }
}
```

Project2Task3Server

```
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Last Modified: October 7, 2022
 * File: RemoteVariableServerUDP.java
 * Part Of: Project2Task3
 * Partial code modified from: https://github.com/CMU-Heinz-95702/Project-
2-Client-Server
 *
 * This Java file acts as a UDP Server which add or subtract the values
 * sent by a client and stores them to a variable 'sum' or gets the current
 * sum of the client. After performing the addition or subtraction
operation
 * from the client, it echoes back the sum to the client. In this example,
the
 * client can make the request from multiple clients (in the form of
different
 * client IDs). Each client has a separate sum variable that gets updated
or
 * returned once the client requests the required operation. The port
number
 * that it would listen to is preset. It creates a DatagramSocket and
 * receives a request from the client in the form of a DatagramPacket. The
 * message from the client is in the form of a byte array which is
converted
 * to int for performing the operation and the sum is displayed to the
user.
 * The sum is then sent to the client. However, if the client sends a
message
 * of "halt!", the server will not halt its execution and will keep the
 * value of the 'sum' variable (for all the clients in the TreeMap) as they
were
 * before the client requested for the halt operation. After the client
resumes,
 * the server will keep updating the value of the previous 'sum' variables
 * for each client as necessary.
 */

// Imports required for UDP/IP, IO operations and TreeMap
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.util.Objects;
import java.util.TreeMap;

public class RemoteVariableServerUDP{

    // Stores the value of the sum to be returned to the client
    static int sum;

    // Create a TreeMap to store each client and its sum
    // Source: https://www.geeksforgeeks.org/treemap-in-java/
    static TreeMap<Integer, Integer> sum_tree_map = new TreeMap<Integer,
Integer>();

    /***
     * The main method for the server to set its port, connect
     * with the client, receive a request from the client, perform
```

```
     * the operation and reply the sum to the client
     * @param args Command line arguments (none here)
     */
    public static void main(String args[]){

        // Prompting the user that the server is running
        System.out.println("\nServer started\n");

        // Hard coded port for the server (as suggest on Piazza)
        int port = 6789;

        // Define a UDP style Datagram socket and set the server port to it
        try (DatagramSocket aSocket = new DatagramSocket(port)) {

            // Keep listening to the client's request
            while (true) {

                // Build a byte array buffer of the maximum size possible
for the request
                // The maximum size is set by
aSocket.getReceiveBufferSize()
                byte[] buffer = new byte[aSocket.getReceiveBufferSize()];

                // Build a DatagramPacket for the request
                DatagramPacket request = new DatagramPacket(buffer,
buffer.length);

                // Block and wait to receive the client request at the port
                aSocket.receive(request);

                // Get the byte array from client's request
                byte[] request_data_bytes = request.getData();

                // Create a new byte array of only the required length
(necessary for the message)
                byte[] new_request_data_bytes = new
byte[request.getLength()];

                // Copy content from data_bytes to new_data_bytes until the
required length
                System.arraycopy(request_data_bytes, 0,
new_request_data_bytes, 0, request.getLength());

                // Stores the user input (operation, operand and client ID)
as a String (from a byte array)
                String user_input = new String(new_request_data_bytes);

                // Split user_input based on a comma separator
                String[] instructions = user_input.split(",");

                // Display visitor ID to the user
                System.out.println("Visitor ID: " + instructions[2]);

                // If client requested for an addition operation
                if (Objects.equals(instructions[0], "1")) {

                    // Display the requested operation to the user
                    System.out.println("Operation Requested: " +
instructions[0] + ". Add");

                    // Perform addition on client's sum and store the
```

```
result into a general sum
                    // variable that would be returned to the client
                    sum = serverAdd(Integer.parseInt(instructions[2]),
Integer.parseInt(instructions[1]));
                }
                // If client requested for a subtraction operation
                else if (Objects.equals(instructions[0], "2")) {

                    // Display the requested operation to the user
                    System.out.println("Operation Requested: " +
instructions[0] + ". Subtract");

                    // Perform subtraction on client's sum and store the
result into a general sum
                    // variable that would be returned to the client
                    sum = serverSubtract(Integer.parseInt(instructions[2]),
Integer.parseInt(instructions[1]));
                }
                // If client requested for a get operation
                else if (Objects.equals(instructions[0], "3")) {

                    // Display the requested operation to the user
                    System.out.println("Operation Requested: " +
instructions[0] + ". Get");

                    // Perform get on client's sum and store the result
into a general sum
                    // variable that would be returned to the client
                    sum = serverGet(Integer.parseInt(instructions[2]));
                }

                // Display the sum to the user and state that the sum is
being returned to the client
                System.out.println("Returning sum of " + sum + " to
client.\n");

                // Convert int sum into byte array
                // Source: https://www.baeldung.com/java-byte-array-to-
number#:~:text=The%20Ints%20class%20also%20has,toByteArray(value)%3B
                int sum_copy = sum;
                byte[] bytes_reply = new byte[Integer.BYTES];
                int length = bytes_reply.length;
                for (int i = 0; i < length; i++) {
                    bytes_reply[length - i - 1] = (byte) (sum_copy & 0xFF);
                    sum_copy >>= 8;
                }

                // Build a DatagramPacket for the reply containing the sum
data after the addition operation
                // The packet includes data in byte array, length of data,
destination address and port
                DatagramPacket reply = new DatagramPacket(bytes_reply,
                        bytes_reply.length, request.getAddress(),
request.getPort());

                // Send a reply to the client (with the value of the
updated sum)
                aSocket.send(reply);
            }
        }
        // Handle socket exceptions
```

```
        catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        }
        // Handle general I/O exceptions
        catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        }
    }

    /***
     * Function to perform the addition operation on the sum variable of
the
     * client who made the request (using the client ID)
     * @param client_id Client ID of the client who made the request
     * @param i Value to add to the sum of the client ID
     * @return Updated value of the sum variable of the client ID
     */
    public static int serverAdd(int client_id, int i) {

        // If TreeMap contains the client ID
        if (sum_tree_map.containsKey(client_id)) {
            // Update the sum of the client by adding i
            sum_tree_map.put(client_id, sum_tree_map.get(client_id) + i);
        }
        // If TreeMap does not contain the client ID
        else {
            // Set the new sum of the client to be i
            sum_tree_map.put(client_id, i);
        }
        // Return the updated sum of the client
        return sum_tree_map.get(client_id);
    }

    /***
     * Function to perform the subtraction operation on the sum variable of
the
     * client who made the request (using the client ID)
     * @param client_id Client ID of the client who made the request
     * @param i Value to subtract from the sum of the client ID
     * @return Updated value of the sum variable of the client ID
     */
    public static int serverSubtract(int client_id, int i) {

        // If TreeMap contains the client ID
        if (sum_tree_map.containsKey(client_id)) {
            // Update the sum of the client by subtracting i
            sum_tree_map.put(client_id, sum_tree_map.get(client_id) - i);
        }
        // If TreeMap does not contain the client ID
        else {
            // Set the new sum of the client to be -i
            sum_tree_map.put(client_id, -i);
        }
        // Return the updated sum of the client
        return sum_tree_map.get(client_id);
    }

    /***
     * Function to get the sum of the client who requested for it (using
the
     * client ID)
```

```
     * @param client_id Client ID of the client who made the request
     * @return Sum variable of the client ID
     */
    public static int serverGet(int client_id) {

        // If TreeMap does not contain the client ID
        if (!sum_tree_map.containsKey(client_id)) {
            // Initialize the sum of the client ID to be 0
            sum_tree_map.put(client_id, 0);
        }
        // Return sum of the client ID
        return sum_tree_map.get(client_id);
    }
}
```

Project2Task3ClientConsole

```
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distributed Systems for ISM\Projects\Project2\Project2Task3\src> java .\RemoteVariableClientUDP.java

The client is running.
Please enter server port: 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:
10
Enter your ID:
100
The result is 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:
4
Enter your ID:
100
The result is 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:
100
The result is 6.
```

```
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:
200
The result is 5.

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:
-3
Enter your ID:
200
The result is 8.
```

```
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:
200
The result is 8.

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:
7
Enter your ID:
300
The result is 7.
```

```
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:
Enter your ID:
300
The result is 10.

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:
300
The result is 10.
```

```
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.
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute
d Systems for ISM\Projects\Project2\Project2Task3\src> java .\RemoteVariabl
eClientUDP.java

The client is running.
Please enter server port: 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:
100
The result is 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:
200
The result is 8.

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:
300
The result is 10.
```

```
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.
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute
d Systems for ISM\Projects\Project2\Project2Task3\src>
```

Project2Task3ServerConsole

```
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute
d Systems for ISM\Projects\Project2\Project2Task3\src> java .\RemoteVariabl
eServerUDP.java

Server started

Visitor ID: 100
Operation Requested: 1. Add
Returning sum of 10 to client.

Visitor ID: 100
Operation Requested: 2. Subtract
Returning sum of 6 to client.

Visitor ID: 100
Operation Requested: 3. Get
Returning sum of 6 to client.

Visitor ID: 200
Operation Requested: 1. Add
Returning sum of 5 to client.
```

```
Visitor ID: 200
Operation Requested: 2. Subtract
Returning sum of 8 to client.

Visitor ID: 200
Operation Requested: 3. Get
Returning sum of 8 to client.

Visitor ID: 300
Operation Requested: 1. Add
Returning sum of 7 to client.

Visitor ID: 300
Operation Requested: 2. Subtract
Returning sum of 10 to client.

Visitor ID: 300
Operation Requested: 3. Get
Returning sum of 10 to client.
```

```
Visitor ID: 100
Operation Requested: 3. Get
Returning sum of 6 to client.

Visitor ID: 200
Operation Requested: 3. Get
Returning sum of 8 to client.

Visitor ID: 300
Operation Requested: 3. Get
Returning sum of 10 to client.
```

Project 2 – Task 4

Project2Task4Client

```java
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Last Modified: October 7, 2022
 * File: RemoteVariableClientTCP.java
 * Part Of: Project2Task4
 * Partial code modified from: https://github.com/CMU-Heinz-95702/Project-2-Client-Server
 *
 * This Java file acts as a TCP Client which sends a request to
 * the server. The request is made of client ID, operation (add,
 * subtract, get) and operand. The operand is added/subtracted
 * from the sum and the updated sum is returned to the client or
 * the sum is directly returned from the server if the client made
 * a get request. The client receives the value of the sum (of the
 * client ID) and prints it to the user. The client sends a packet
 * to the server and waits for the server to execute the requested
 * operation. When the response packet arrives from the server, the
 * client creates an int object and displays the output to the user.
 * If the client wishes to halt its execution, the server will not
 * be affected.
 */

// imports required for TCP/IP and IO operations
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class RemoteVariableClientTCP {

    // Stores the value of port number of the server
    static int serverPort;

    /***
     * The main method for the client to request for the server port from the user,
     * request a message (client ID, operation, operand) from the user, send a request
     * message to the server, receive a reply from the server (sum of the client ID)
     * and print the reply to the user
     * @param args Command line arguments (none here)
     */
    public static void main(String args[]) throws IOException {

        // Prompting the user that the client is running
        System.out.println("\nThe client is running.");

        // Creating a Scanner object for taking inputs from the user
        Scanner s = new Scanner(System.in);

        // Requesting the user for the server side port
        System.out.print("Please enter server port: ");

        // Getting the server port from the user
        serverPort = s.nextInt();
```

```
        System.out.println();

        // Stores the input from the user
        String user_input;

        // Create a BufferReader object to get input from the user
        BufferedReader typed = new BufferedReader(new
InputStreamReader(System.in));

        // Keeping executing until the client is terminated
        while (true) {

            // Initialize user input to null
            user_input = "";

            // Prompt the user for operation
            System.out.println("""
                    1. Add a value to your sum.
                    2. Subtract a value from your sum.
                    3. Get your sum.
                    4. Exit client""");

            // Update user input
            user_input = user_input + typed.readLine();

            // Switch case for user input
            switch (user_input) {

                // If user requested for an addition operation
                case "1" -> {

                    // Prompt the user for an operand
                    System.out.println("Enter value to add:");

                    // Update user input
                    user_input = user_input + "," + typed.readLine() + ",";
                }

                // If user requested for a subtraction operation
                case "2" -> {
                    // Prompt the user for an operand
                    System.out.println("Enter value to subtract:");

                    // Update user input
                    user_input = user_input + "," + typed.readLine() + ",";
                }

                // If user requested for a get operation
                case "3" -> user_input = user_input + ",,";

                // If user requested for halting
                case "4" -> {
                    // Prompt the user that the client is quitting
                    System.out.println("Client side quitting. The remote
variable server is still running.");
                    // Halt client execution
                    System.exit(0);
                }
            }

            // Prompt user for client ID
```

```
            System.out.println("Enter your ID:");

            // Update user input
            user_input = user_input + typed.readLine();

            // Request the operation from server and store the value of sum
            int serverSumReturned = operations(user_input);

            // Display the sum received from the server to the user
            System.out.println("The result is " + serverSumReturned +
".\n");
        }
    }

    /***
     * Function to communicate with the server and perform the required
     * operation on the requested integer value by the client
     * @param user_input Input from the user containing client ID,
operation and operand
     * @return Updated sum from the server
     */
    public static int operations(String user_input) {

        // Define a TCP style Socket
        Socket clientSocket = null;

        // Stores the sum returned from the server
        int serverSumReturned = 0;

        try {
            // Creating a String object for localhost
            String localhost = "";

            // Updating clientSocket with localhost and the server port
            clientSocket = new Socket(localhost, serverPort);

            // Set up "in" to read from the server socket
            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

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

            // Request to the server with the user_input
            out.println(user_input);

            // Flush to server socket
            out.flush();

            // Store the sum returned from the server and parse it to
integer
            serverSumReturned = Integer.parseInt(in.readLine()); // read a
line of data from the stream
        }
        // Handle general I/O exceptions
        catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        }
        // Always close the socket
        finally {
```

```
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }

        // Return the updated sum
        return serverSumReturned;
    }
}
```

Project2Task4Server

```java
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Last Modified: October 7, 2022
 * File: RemoteVariableServerTCP.java
 * Part Of: Project2Task4
 * Partial code modified from: https://github.com/CMU-Heinz-95702/Project-2-Client-Server
 *
 * This Java file acts as a TCP Server which add or subtract the values
 * sent by a client and stores them to a variable 'sum' or gets the current
 * sum of the client. After performing the addition or subtraction operation
 * from the client, it echoes back the sum to the client. In this example, the
 * client can make the request from multiple clients (in the form of different
 * client IDs). Each client has a separate sum variable that gets updated or
 * returned once the client requests the required operation. The port number
 * that it would listen to is preset. It creates a Socket and receives a request
 * from the client. The message from the client is in the form of String which is
 * processed to perform the necessary operations. The sum for the client is then
 * sent to the client. However, if the client request to halt, the server will not
 * halt its execution and will keep the value of the 'sum' variable (for all the
 * clients in the TreeMap) as they were before the client requested for the halt
 * operation.
 */

// Imports required for TCP/IP, IO operations and TreeMap
import java.net.*;
import java.io.*;
import java.util.Objects;
import java.util.Scanner;
import java.util.TreeMap;

public class RemoteVariableServerTCP {

    // Stores the value of the sum to be returned to the client
    static int sum;

    // Create a TreeMap to store each client and its sum
    // Source: https://www.geeksforgeeks.org/treemap-in-java/
    static TreeMap<Integer, Integer> sum_tree_map = new TreeMap<Integer, Integer>();

    /***
     * The main method for the server to set its port, connect
     * with the client, receive a request from the client, perform
     * the operation and reply the sum to the client
     * @param args Command line arguments (none here)
     */
```

```
    public static void main(String args[]) {

        // Prompting the user that the server is running
        System.out.println("\nServer started\n");

        // Define a TCP style Socket
        Socket clientSocket = null;

        // Define a TCP style ServerSocket
        ServerSocket listenSocket;
        try {

            // Hard coded port for the server (as suggest on Piazza)
            int serverPort = 6789;

            // Create a new server socket
            listenSocket = new ServerSocket(serverPort);

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

                // Get the input (operation, operand and client ID) from
the client and store it in a String object
                String user_input = in.nextLine();

                // Split user_input based on a comma separator
                String[] instructions = user_input.split(",");

                // Display visitor ID to the user
                System.out.println("Visitor ID: " + instructions[2]);

                // If client requested for an addition operation
                if (Objects.equals(instructions[0], "1")) {

                    // Display the requested operation to the user
                    System.out.println("Operation Requested: " +
instructions[0] + ". Add");
```

```
                    // Perform addition on client's sum and store the
result into a general sum
                    // variable that would be returned to the client
                    sum = serverAdd(Integer.parseInt(instructions[2]),
Integer.parseInt(instructions[1]));
                }
                // If client requested for a subtraction operation
                else if (Objects.equals(instructions[0], "2")) {

                    // Display the requested operation to the user
                    System.out.println("Operation Requested: " +
instructions[0] + ". Subtract");

                    // Perform subtraction on client's sum and store the
result into a general sum
                    // variable that would be returned to the client
                    sum = serverSubtract(Integer.parseInt(instructions[2]),
Integer.parseInt(instructions[1]));
                }
                // If client requested for a get operation
                else if (Objects.equals(instructions[0], "3")) {

                    // Display the requested operation to the user
                    System.out.println("Operation Requested: " +
instructions[0] + ". Get");

                    // Perform get on client's sum and store the result
into a general sum
                    // variable that would be returned to the client
                    sum = serverGet(Integer.parseInt(instructions[2]));
                }

                // Display the sum to the user and state that the sum is
being returned to the client
                System.out.println("Returning sum of " + sum + " to
client.\n");

                // Reply the sum to the client
                out.println(sum);

                // Flush to client socket
                out.flush();
            }
        }
        // Handle IO exceptions
        catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
            // If quitting (typically by you sending quit signal) clean up
sockets
        }
        // Always close the socket
        finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }
```

```java
    /***
     * Function to perform the addition operation on the sum variable of
the
     * client who made the request (using the client ID)
     * @param client_id Client ID of the client who made the request
     * @param i Value to add to the sum of the client ID
     * @return Updated value of the sum variable of the client ID
     */
    public static int serverAdd(int client_id, int i) {

        // If TreeMap contains the client ID
        if (sum_tree_map.containsKey(client_id)) {
            // Update the sum of the client by adding i
            sum_tree_map.put(client_id, sum_tree_map.get(client_id) + i);
        }
        // If TreeMap does not contain the client ID
        else {
            // Set the new sum of the client to be i
            sum_tree_map.put(client_id, i);
        }
        // Return the updated sum of the client
        return sum_tree_map.get(client_id);
    }

    /***
     * Function to perform the subtraction operation on the sum variable of
the
     * client who made the request (using the client ID)
     * @param client_id Client ID of the client who made the request
     * @param i Value to subtract from the sum of the client ID
     * @return Updated value of the sum variable of the client ID
     */
    public static int serverSubtract(int client_id, int i) {

        // If TreeMap contains the client ID
        if (sum_tree_map.containsKey(client_id)) {
            // Update the sum of the client by subtracting i
            sum_tree_map.put(client_id, sum_tree_map.get(client_id) - i);
        }
        // If TreeMap does not contain the client ID
        else {
            // Set the new sum of the client to be -i
            sum_tree_map.put(client_id, -i);
        }
        // Return the updated sum of the client
        return sum_tree_map.get(client_id);
    }

    /***
     * Function to get the sum of the client who requested for it (using
the
     * client ID)
     * @param client_id Client ID of the client who made the request
     * @return Sum variable of the client ID
     */
    public static int serverGet(int client_id) {

        // If TreeMap does not contain the client ID
        if (!sum_tree_map.containsKey(client_id)) {
            // Initialize the sum of the client ID to be 0
```

```
            sum_tree_map.put(client_id, 0);
        }
        // Return sum of the client ID
        return sum_tree_map.get(client_id);
    }
}
```

Project2Task4ClientConsole

```
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distributed Systems for ISM\Projects\Project2\Project2Task4\src> java .\RemoteVariableClientTCP.java

The client is running.
Please enter server port: 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:
1
Enter your ID:
400
The result is 1.
```

```
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:
-1
Enter your ID:
400
The result is 2.

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:
400
The result is 2.
```

```
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:
3
Enter your ID:
500
The result is 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:
1
Enter your ID:
500
The result is 2.
```

```
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:
500
The result is 2.

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:
4
Enter your ID:
600
The result is 4.
```

```
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:
Enter your ID:
600
The result is 4.

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:
600
The result is 4.
```

```
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.
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute
d Systems for ISM\Projects\Project2\Project2Task4\src> java .\RemoteVariabl
eClientTCP.java

The client is running.
Please enter server port: 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:
400
The result is 2.
```

```
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:
500
The result is 2.

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:
600
The result is 4.
```

```
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.
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute
d Systems for ISM\Projects\Project2\Project2Task4\src>
```

Project2Task4ServerConsole

```
PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute
d Systems for ISM\Projects\Project2\Project2Task4\src> java .\RemoteVariabl
eServerTCP.java

Server started

Visitor ID: 400
Operation Requested: 1. Add
Returning sum of 1 to client.

Visitor ID: 400
Operation Requested: 2. Subtract
Returning sum of 2 to client.

Visitor ID: 400
Operation Requested: 3. Get
Returning sum of 2 to client.

Visitor ID: 500
Operation Requested: 1. Add
Returning sum of 3 to client.
```

```
Visitor ID: 500
Operation Requested: 2. Subtract
Returning sum of 2 to client.

Visitor ID: 500
Operation Requested: 3. Get
Returning sum of 2 to client.

Visitor ID: 600
Operation Requested: 1. Add
Returning sum of 4 to client.

Visitor ID: 600
Operation Requested: 2. Subtract
Returning sum of 4 to client.

Visitor ID: 600
Operation Requested: 3. Get
Returning sum of 4 to client.
```

```
Visitor ID: 400
Operation Requested: 3. Get
Returning sum of 2 to client.

Visitor ID: 500
Operation Requested: 3. Get
Returning sum of 2 to client.

Visitor ID: 600
Operation Requested: 3. Get
Returning sum of 4 to client.
```

Project 2 – Task 5

Project2Task5Client

```
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Last Modified: October 7, 2022
 * File: SigningClientTCP.java
 * Part Of: Project2Task5
 * Partial code modified from: https://github.com/CMU-Heinz-95702/Project-
2-Client-Server
 *
 * This Java file acts as a TCP Client which sends a request to
 * the server. The request is made of client ID, operation (add,
 * subtract, get) and operand. The operand is added/subtracted
 * from the sum and the updated sum is returned to the client or
 * the sum is directly returned from the server if the client made
 * a get request. The client receives the value of the sum (of the
 * client ID) and prints it to the user. The client sends a packet
 * to the server and waits for the server to execute the requested
 * operation. When the response packet arrives from the server, the
 * client creates an int object and displays the output to the user.
 * If the client wishes to halt its execution, the server will not
 * be affected.
 * The client will send a signed request. Each time the client
 * program runs, it will create new RSA public and private keys and
 * display these keys to the user. The client's ID will be formed by
 * taking the least significant 20 bytes of the hash of the client's
 * public key. The client will also transmit its public key with each
 * request. Finally, the client will sign each request.
 */

// imports required for TCP/IP, IO operations, BigInteger, RSA256, List and
Random
import java.io.*;
import java.math.BigInteger;
import java.net.Socket;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.Scanner;

public class SigningClientTCP {

    // Stores the value of port number of the server
    static int serverPort;

    /***
     * The main method for the client to request for the server port from
the user,
     * request a message (operation, operand) from the user, send a request
     * message to the server, receive a reply from the server (sum of the
client ID)
     * and print the reply to the user. The client's ID will be formed by
taking the
     * least significant 20 bytes of the hash of the client's public key.
The client
```

```
     * will also transmit its public key with each request. Finally, the
client will
     * sign each request.
     * @param args Command line arguments (none here)
     */
    public static void main(String args[]) throws Exception {

        // Prompting the user that the client is running
        System.out.println("\nThe client is running.");

        // Creating a Scanner object for taking inputs from the user
        Scanner s = new Scanner(System.in);

        // Requesting the user for the server side port
        System.out.print("Please enter server port: ");

        // Getting the server port from the user
        serverPort = s.nextInt();
        System.out.println();

        // List to store the generated parts of the RSA public and private
keys
        List keys_list = generateRSAKeys();

        BigInteger e; // e is the exponent of the public key
        BigInteger d; // d is the exponent of the private key
        BigInteger n; // n is the modulus for both the private and public
keys
        BigInteger public_key; // Stores the RSA public key

        // Populate the values of e, d, n and public key from the keys_list
        e = (BigInteger) keys_list.get(0);
        d = (BigInteger) keys_list.get(1);
        n = (BigInteger) keys_list.get(2);
        public_key = new BigInteger(e + String.valueOf(n));

        // Compute SHA-256 hash of the public key
        byte[] hash_of_public_key =
computeSHA256(String.valueOf(public_key));

        // Stores the least significant 20 bytes of the hash
        byte[] client_LSB_20_bytes = new byte[20];

        // Populate the client_LSB_20_bytes byte array
        for (int i = 0; i < 20; i++) {
            client_LSB_20_bytes[i] =
hash_of_public_key[hash_of_public_key.length - 1 - i];
        }

        // Computes the client ID by converting client_LSB_20_bytes bytes
array to a hexadecimal String
        String client_ID = bytesToHex(client_LSB_20_bytes);

        // Stores user input
        String user_input;

        // Create a BufferReader object to get input from the user
        BufferedReader typed = new BufferedReader(new
InputStreamReader(System.in));

        // Keeping executing until the client is terminated
```

```
        while (true) {

            // Initialize user input to null
            user_input = "";

            // Prompt the user for operation
            System.out.println("""
                    1. Add a value to your sum.
                    2. Subtract a value from your sum.
                    3. Get your sum.
                    4. Exit client""");

            // Read operation from user
            String user_operation = typed.readLine();

            // Stores operand for the operation
            String operand = "";

            // Update user input
            user_input = user_input + user_operation;

            // Switch case for user input
            switch (user_input) {

                // If user requested for an addition operation
                case "1" -> {

                    // Prompt the user for an operand
                    System.out.println("Enter value to add:");
                    operand = typed.readLine();

                    // Update user input
                    user_input = user_input + "," + operand + ",";
                }

                // If user requested for a subtraction operation
                case "2" -> {

                    // Prompt the user for an operand
                    System.out.println("Enter value to subtract:");
                    operand = typed.readLine();

                    // Update user input
                    user_input = user_input + "," + operand + ",";
                }

                // If user requested for a get operation
                case "3" -> user_input = user_input + ",,";

                // If user requested for halting
                case "4" -> {
                    // Prompt the user that the client is quitting
                    System.out.println("Client side quitting. The remote
variable server is still running.");
                    // Halt client execution
                    System.exit(0);
                }
            }

            // Stores client request without signature
            String client_request_without_sign = client_ID + "," + e + ","
```

```
+ n + "," +
                    user_operation + "," + operand;

            // Generates signature for client request
            String signature = sign(client_request_without_sign, d, n);

            // Stores client request with signature
            String client_request_with_sign = client_request_without_sign +
"," + signature;

            // Request the operation from server and store the value of sum
            int serverSumReturned = operations(client_request_with_sign);

            // Display the sum received from the server to the user
            System.out.println("The result is " + serverSumReturned +
".\n");
        }
    }

    /***
     * Function to communicate with the server and perform the required
     * operation on the requested integer value by the client
     * @param client_request_with_sign Input from the user containing
client ID, operation and operand
     * @return Updated sum from the server
     */
    public static int operations(String client_request_with_sign) {

        // Define a TCP style Socket
        Socket clientSocket = null;

        // Stores the sum returned from the server
        int serverSumReturned = 0;

        try {
            // Creating a String object for localhost
            String localhost = "";

            // Updating clientSocket with localhost and the server port
            clientSocket = new Socket(localhost, serverPort);

            // Set up "in" to read from the server socket
            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

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

            // Request to the server with the user_input
            out.println(client_request_with_sign);

            // Flush to server socket
            out.flush();

            // Store the sum returned from the server and parse it to
integer
            serverSumReturned = Integer.parseInt(in.readLine()); // read a
line of data from the stream
        }
        // Handle general I/O exceptions
```

```
        catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        }
        // Always close the socket
        finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }

        // Return the updated sum
        return serverSumReturned;
    }

    /***
     * Function to generate RSA public and private keys
     * @return A List of (e, d, n) which are the components of the public
and private keys
     */
    // Source to generate RSA public and private keys - CMU Heinz 95-702 -
Project 2 GitHub Page
    // https://github.com/CMU-Heinz-95702/Project-2-Client-Server
    public static List generateRSAKeys() {
        List<BigInteger> keys_list = new ArrayList<>();

        // Each public and private key consists of an exponent and a
modulus
        BigInteger n; // n is the modulus for both the private and public
keys
        BigInteger e; // e is the exponent of the public key
        BigInteger d; // d is the exponent of the private key

        // Generate a random number
        Random rnd = new Random();

        // Step 1: Generate two large random primes.
        BigInteger p = new BigInteger(2048, 100, rnd);
        BigInteger q = new BigInteger(2048, 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("65537");

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

        // Displaying the constituents of client's public and private keys
        System.out.println(" e = " + e);
```

```
        System.out.println(" d = " + d);
        System.out.println(" n = " + n);

        // Display RSA public and RSA private keys to the user
        System.out.println("\nRSA Public Key (e, n) = (" + e + ", \n" + n +
")\n");
        System.out.println("RSA Private Key (d, n) = (" + d + ", \n" + n +
")\n");

        // Add e, d and n to the list
        keys_list.add(e);
        keys_list.add(d);
        keys_list.add(n);

        // Return list of the components of the RSA keys
        return keys_list;
    }

    /***
     * Computes the SHA256 hash value of the string passed into the
function
     * @param input String input whose hash value is to be computed
     * @return A byte array with the SHA256 hash of the input String
     */
    // Source: Shivam Patel Project 1 Task 1 - CMU Heinz 95-702
    public static byte[] computeSHA256(String input) {

        // Source: CMU 95702 Fall 2022 Lab1-InstallationAndRaft Code
        byte[] digest = new byte[0];
        try {
            // Access MessageDigest class for SHA256
            MessageDigest md = MessageDigest.getInstance("SHA-256");

            // Compute the digest
            md.update(input.getBytes());

            // Store digest as a byte array for further use
            digest = md.digest();
        }
        // Handles No SHA-256 Algorithm exceptions
        catch (NoSuchAlgorithmException e) {
            // Print error message in console
            System.out.println("No SHA-256 available" + e);
        }
        // Return the SHA256 hash of input in byte[] form
        return digest;
    }

    /**
     * Function to sign a message using RSA private key.
     * Signing proceeds as follows:
     * 1) Get the bytes from the string to be signed.
     * 2) Compute SHA-256 digest of these bytes.
     * 3) Copy these bytes into a byte array that is one byte longer than
needed.
     *    The resulting byte array has its extra byte set to zero. This is
because
     *    RSA works only on positive numbers. The most significant byte (in
the
     *    new byte array) is the 0'th byte. It must be set to zero.
     * 4) Create a BigInteger from the byte array.
```

```
     * 5) Encrypt the BigInteger with RSA d and n.
     * 6) Return to the caller a String representation of this BigInteger.
     * @param message a sting to be signed
     * @param d BigInteger d, part of the RSA Private Key
     * @param n BigInteger n, part of the RSA Private Key
     * @return String representing a big integer - the encrypted hash.
     */
    // Source to generate RSA public and private keys - CMU Heinz 95-702 -
Project 2 GitHub Page
    // https://github.com/CMU-Heinz-95702/Project-2-Client-Server
    public static String sign(String message, BigInteger d, BigInteger n) {

        // Compute the digest with SHA-256
        byte[] bigDigest = computeSHA256(message);

        // We add a 0 byte as the most significant byte to keep
        // the value to be signed non-negative.
        byte[] messageDigest = new byte[bigDigest.length + 1];
        messageDigest[0] = 0;   // most significant set to 0

        System.arraycopy(bigDigest, 0, messageDigest, 1, bigDigest.length +
1 - 1);

        // From the digest, create a BigInteger
        BigInteger m = new BigInteger(messageDigest);

        // encrypt the digest with the private key
        BigInteger c = m.modPow(d, n);

        // return this as a big integer string
        return c.toString();
    }

    // Code to convert from byte array to hexadecimal String
    // Source: https://stackoverflow.com/questions/9655181/how-to-convert-
a-byte-array-to-a-hex-string-in-java
    private static final char[] HEX_ARRAY =
"0123456789ABCDEF".toCharArray();

    /***
     * Function to convert a byte array to hexadecimal String
     * @param bytes Byte array to be converted to hexadecimal String
     * @return Hexadecimal notation (in String form) of the input byte
array
     */
    public static String bytesToHex(byte[] bytes) {
        char[] hexChars = new char[bytes.length * 2];
        for (int j = 0; j < bytes.length; j++) {
            int v = bytes[j] & 0xFF;
            hexChars[j * 2] = HEX_ARRAY[v >>> 4];
            hexChars[j * 2 + 1] = HEX_ARRAY[v & 0x0F];
        }
        return new String(hexChars);
    }
}
```

Project2Task5Server

```java
/**
 * Author: Shivam Patel
 * Andrew ID: shpatel
 * Last Modified: October 7, 2022
 * File: VerifyingServerTCP.java
 * Part Of: Project2Task5
 * Partial code modified from: https://github.com/CMU-Heinz-95702/Project-2-Client-Server
 *
 * This Java file acts as a TCP Server which add or subtract the values
 * sent by a client and stores them to a variable 'sum' or gets the current
 * sum of the client. After performing the addition or subtraction operation
 * from the client, it echoes back the sum to the client. In this example, the
 * client can make the request from multiple clients (in the form of different
 * client IDs). Each client has a separate sum variable that gets updated or
 * returned once the client requests the required operation. The port number
 * that it would listen to is preset. It creates a Socket and receives a request
 * from the client. The message from the client is in the form of String which is
 * processed to perform the necessary operations. The sum for the client is then
 * sent to the client. However, if the client request to halt, the server will not
 * halt its execution and will keep the value of the 'sum' variable (for all the
 * clients in the TreeMap) as they were before the client requested for the halt
 * operation.
 * The server will make two checks before servicing any client request. First,
 * does the public key (included with each request) hash to the ID (also provided
 * with each request)? Second, is the request properly signed? If both of these
 * are true, the request is carried out on behalf of the client. The server will add,
 * subtract or get. Otherwise, the server returns the message "Error in request".
 */

// Imports required for TCP/IP, IO operations, TreeMap, BigInteger, SHA256, Arrays
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.math.BigInteger;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Objects;
```

```java
import java.util.Scanner;
import java.util.TreeMap;

public class VerifyingServerTCP {

    // Stores the value of the sum to be returned to the client
    static int sum;

    // Create a TreeMap to store each client and its sum
    // Source: https://www.geeksforgeeks.org/treemap-in-java/
    static TreeMap<String, Integer> sum_tree_map = new TreeMap<String,
Integer>();

    /***
     * The main method for the server to set its port, connect
     * with the client, receive a request from the client, perform
     * the operation and reply the sum to the client
     * @param args Command line arguments (none here)
     */
    public static void main(String args[]) {

        // Prompting the user that the server is running
        System.out.println("\nServer started\n");

        // Define a TCP style Socket
        Socket clientSocket = null;

        // Define a TCP style ServerSocket
        ServerSocket listenSocket;
        try {

            // Hard coded port for the server (as suggest on Piazza)
            int serverPort = 6789;

            // Create a new server socket
            listenSocket = new ServerSocket(serverPort);

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

                // Get the input (Client ID, e, n, operation, operand,
signature) from the client and store it in a String object
                String user_input = in.nextLine();

                // Split user_input based on a comma separator
                String[] request_split = user_input.split(",");

                // Use the request_split to populate the values for client
ID, e, n, user_operation,
                // operand, and encryptedHashStr
                String client_ID = request_split[0];
                BigInteger e = new BigInteger(request_split[1]);
                BigInteger n = new BigInteger(request_split[2]);
                String user_operation = request_split[3];
                String operand = request_split[4];
                String encryptedHashStr = request_split[5];

                System.out.println("Visitor's Public Key Material: ");
                System.out.println(" e = " + e);
                System.out.println(" n = " + n);

                System.out.println("\nVisitor's Public Key (e, n) = (" + e
+ ", \n" + n + ")\n");

                // Stores the result of the public key and client ID
verification
                boolean public_key_client_ID_verification_result;

                // Stores the result of signature verification
                boolean signature_verification_result;

                // Generate the message to check
                String messageToCheck = request_split[0] + "," +
request_split[1] + "," +
                        request_split[2] + "," + request_split[3] + "," +
request_split[4];

                // Verify signature
                signature_verification_result =
verifySignature(messageToCheck, encryptedHashStr, e, n);
                if (signature_verification_result) {
                    System.out.println("Signature verified!");
                }
                else {
                    System.out.println("Signature not verified!");
                }

                // Verify public key hash client ID
                public_key_client_ID_verification_result =
verifyPublicKeyClientID(client_ID, e, n);
                if (public_key_client_ID_verification_result) {
                    System.out.println("\nPublic Key hash Client ID
verified!\n");
                }
                else {
                    System.out.println("\nPublic Key hash Client ID not
verified!\n");
                }
```

```
                // If both the above verifications are correct
                if (signature_verification_result &&
public_key_client_ID_verification_result) {

                    // Display visitor ID to the user
                    System.out.println("Visitor ID: " + client_ID);

                    // If client requested for an addition operation
                    if (Objects.equals(user_operation, "1")) {

                        // Display the requested operation to the user
                        System.out.println("Operation Requested: " +
user_operation + ". Add");

                        // Perform addition on client's sum and store the
result into a general sum
                        // variable that would be returned to the client
                        sum = serverAdd(client_ID,
Integer.parseInt(operand));
                    }
                    // If client requested for a subtraction operation
                    else if (Objects.equals(user_operation, "2")) {

                        // Display the requested operation to the user
                        System.out.println("Operation Requested: " +
user_operation + ". Subtract");

                        // Perform subtraction on client's sum and store
the result into a general sum
                        // variable that would be returned to the client
                        sum = serverSubtract(client_ID,
Integer.parseInt(operand));
                    }
                    // If client requested for a get operation
                    else if (Objects.equals(user_operation, "3")) {

                        // Display the requested operation to the user
                        System.out.println("Operation Requested: " +
user_operation + ". Get");

                        // Perform get on client's sum and store the result
into a general sum
                        // variable that would be returned to the client
                        sum = serverGet(client_ID);
                    }

                    // Display the sum to the user and state that the sum
is being returned to the client
                    System.out.println("Returning sum of " + sum + " to
client.\n");

                    // Reply the sum to the client
                    out.println(sum);

                    // Flush to client socket
                    out.flush();
                }
                // If one or both the above verifications are not correct
                else {
                    // Display error message to the user
                    System.out.println("Error in request");
```

```
                }

            }
        }
        // Handle IO exceptions
        catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
            // If quitting (typically by you sending quit signal) clean up
sockets
        }
        // Handle RuntimeException exceptions
        catch (Exception e) {
            throw new RuntimeException(e);
        }
        // Always close the socket
        finally {
            try {
                if (clientSocket != null) {
                    clientSocket.close();
                }
            } catch (IOException e) {
                // ignore exception on close
            }
        }
    }

    /***
     * Function to perform the addition operation on the sum variable of
the
     * client who made the request (using the client ID)
     * @param client_id Client ID of the client who made the request
     * @param i Value to add to the sum of the client ID
     * @return Updated value of the sum variable of the client ID
     */
    public static int serverAdd(String client_id, int i) {

        // If TreeMap contains the client ID
        if (sum_tree_map.containsKey(client_id)) {
            // Update the sum of the client by adding i
            sum_tree_map.put(client_id, sum_tree_map.get(client_id) + i);
        }
        // If TreeMap does not contain the client ID
        else {
            // Set the new sum of the client to be i
            sum_tree_map.put(client_id, i);
        }
        // Return the updated sum of the client
        return sum_tree_map.get(client_id);
    }

    /***
     * Function to perform the subtraction operation on the sum variable of
the
     * client who made the request (using the client ID)
     * @param client_id Client ID of the client who made the request
     * @param i Value to subtract from the sum of the client ID
     * @return Updated value of the sum variable of the client ID
     */
    public static int serverSubtract(String client_id, int i) {

        // If TreeMap contains the client ID
```

```
        if (sum_tree_map.containsKey(client_id)) {
            // Update the sum of the client by subtracting i
            sum_tree_map.put(client_id, sum_tree_map.get(client_id) - i);
        }
        // If TreeMap does not contain the client ID
        else {
            // Set the new sum of the client to be -i
            sum_tree_map.put(client_id, -i);
        }
        // Return the updated sum of the client
        return sum_tree_map.get(client_id);
    }

    /***
     * Function to get the sum of the client who requested for it (using
the
     * client ID)
     * @param client_id Client ID of the client who made the request
     * @return Sum variable of the client ID
     */
    public static int serverGet(String client_id) {

        // If TreeMap does not contain the client ID
        if (!sum_tree_map.containsKey(client_id)) {
            // Initialize the sum of the client ID to be 0
            sum_tree_map.put(client_id, 0);
        }
        // Return sum of the client ID
        return sum_tree_map.get(client_id);
    }

    /**
     * Function to verify a signature. The verifying proceeds as follows:
     * 1) Decrypt the encryptedHash to compute a decryptedHash
     * 2) Hash the messageToCheck using SHA-256 (be sure to handle
     *    the extra byte as described in the signing method.)
     * 3) If this new hash is equal to the decryptedHash, return true else
false.
     *
     * @param messageToCheck  a normal string that needs to be verified.
     * @param encryptedHashStr integer string - possible evidence attesting
to its origin.
     * @return true or false depending on whether the verification was a
success
     */
    // Source to generate RSA public and private keys - CMU Heinz 95-702 -
Project 2 GitHub Page
    // https://github.com/CMU-Heinz-95702/Project-2-Client-Server
    public static boolean verifySignature(String messageToCheck, String
encryptedHashStr, BigInteger e, BigInteger n) {

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

        // Decrypt it
        BigInteger decryptedHash = encryptedHash.modPow(e, n);

        // Compute SHA256 hash of the messageToCheck
        byte[] messageToCheckDigest = computeSHA256(messageToCheck);

        // messageToCheckDigest is a full SHA-256 digest
```

```
        // add a zero byte as the most significant byte to keep
        // the value to be signed non-negative.
        byte[] messageToCheckDigestWithExtraByte = new
byte[messageToCheckDigest.length + 1];
        messageToCheckDigestWithExtraByte[0] = 0;

        // Generate the messageToCheckDigestWithExtraByte byte array
        System.arraycopy(messageToCheckDigest, 0,
messageToCheckDigestWithExtraByte, 1, messageToCheckDigest.length + 1 - 1);

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

        // Inform the user on how the two compare (true or false)
        return bigIntegerToCheck.compareTo(decryptedHash) == 0;
    }

    /***
     * Function to verify if the client ID hashes to the public key
     * @param client_ID Client ID of the client
     * @param e BigInteger e, which forms the public key
     * @param n BigInteger n, which forms the public key
     * @return Result of verification in terms of true or false
     */
    public static boolean verifyPublicKeyClientID(String client_ID,
BigInteger e, BigInteger n) {

        // Generate the public key by concatenating e with n
        BigInteger public_key = new BigInteger(e + String.valueOf(n));

        // Compute SHA256 hash of the public key and store it to a byte
array
        byte[] hash_of_public_key =
computeSHA256(String.valueOf(public_key));

        // Stores the least significant 20 bytes of the hash
        byte[] client_LSB_20_bytes = new byte[20];

        // Populate the client_LSB_20_bytes byte array
        for (int i = 0; i < 20; i++) {
            client_LSB_20_bytes[i] =
hash_of_public_key[hash_of_public_key.length - 1 - i];
        }

        // Compute the expected client ID by converting the least
significant 20 bytes to hexadecimal String
        String expected_client_ID = bytesToHex(client_LSB_20_bytes);

        // Verify if the expected client ID matches the actual client ID
        return expected_client_ID.equals(client_ID);
    }

    /***
     * Computes the SHA256 hash value of the string passed into the
function
     * @param input String input whose hash value is to be computed
     * @return A byte array with the SHA256 hash of the input String
     */
    // Source: Shivam Patel Project 1 Task 1 - CMU Heinz 95-702
    public static byte[] computeSHA256(String input) {
```

```
        // Source: CMU 95702 Fall 2022 Lab1-InstallationAndRaft Code
        byte[] digest = new byte[0];
        try {
            // Access MessageDigest class for SHA256
            MessageDigest md = MessageDigest.getInstance("SHA-256");

            // Compute the digest
            md.update(input.getBytes());

            // Store digest as a byte array for further use
            digest = md.digest();
        }
        // Handles No SHA-256 Algorithm exceptions
        catch (NoSuchAlgorithmException e) {
            // Print error message in console
            System.out.println("No SHA-256 available" + e);
        }
        // Return the SHA256 hash of input in byte[] form
        return digest;
    }

    // Code to convert from byte array to hexadecimal String
    // Source: https://stackoverflow.com/questions/9655181/how-to-convert-
a-byte-array-to-a-hex-string-in-java
    private static final char[] HEX_ARRAY =
"0123456789ABCDEF".toCharArray();

    /***
     * Function to convert a byte array to hexadecimal String
     * @param bytes Byte array to be converted to hexadecimal String
     * @return Hexadecimal notation (in String form) of the input byte
array
     */
    public static String bytesToHex(byte[] bytes) {
        char[] hexChars = new char[bytes.length * 2];
        for (int j = 0; j < bytes.length; j++) {
            int v = bytes[j] & 0xFF;
            hexChars[j * 2] = HEX_ARRAY[v >>> 4];
            hexChars[j * 2 + 1] = HEX_ARRAY[v & 0x0F];
        }
        return new String(hexChars);
    }
}
```

Project2Task5ClientConsole

Note: I am pasting the output of my Client Console here, because taking multiple screenshots of such a huge output (displaying the interaction of 3 different clients) was not seeming the best possible representation. My comments in the output are mentioned in orange for better understanding of the grader.

(Client 1 Console Output)

PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute

d Systems for ISM\Projects\Project2\Project2Task5\src> java .\SigningClient

TCP.java

The client is running.

Please enter server port: 6789

e = 65537

d = 157708540975945895842783457264519813365323281037373427623954935118017486332535292017112521940433391519903347523102350853851989589666738165686951036779761304021323018620751405431494725505160052101430710264322215726428255287719158582061319439010747260412172197014690554495216777518174229035906738740440635765759003849733835526452385174720071552356753125060352415696624036928918475740442929322899938053825037869209306284205487540723664501839944400472180308921598782077778935102314958420020660370631941090811355389808906258202431122781292786765640585340499483909492031465659519540572281932892925100081480547251101608532703907723975943548010907170846041396300790944007263531275077394001563683854477675445001518807877398953287239936988778077566588741396137317924506608042320419690922785661205861043525176748522904641420821467732721847003178021068091950886997069236286581172178694101022018919025268943567879574781610106532861803034442717513234055733277557237352417491749905038262749937129927347270935239651383456616721807861526750196105703336174845736424427939478272461610222929625601323309806504503124514664950454460456549262115510983355675844123575706755006270263885770161780768757147895497181667524073603534015946937887893791644218915121

n = 42271255367635541187879839019855363823660348735619575175621179431636791958510348995646408533025983313729114091945355068949727380221622915891479735378656155501392362976353461444373930822590791111085290820796225316195364470946998061398522318410074611286378999941727433928890523693135595469822174703825029431027556481534523655053412814181953005303687086651181204316836323701323177557569867322542608646818254905052432644668827601964773630759823468053434546259027521905162971125109056602891063836694227325133956885942931943248998930488395506794240673352665799668628601145821962014357227275105327521508486322134863384394948361106507741132455125748786430080919903269647252827619873944211366707980562217159119452212204820439413387018768006038965015841972934541108053411291264066374563172111120511827609918519680255080737524333457460087811553339100881194809160812923177944538334528729646498631224969566513172577791133448203900526047483405919900284775721813452033616829108622556784271470658531480972801105408194125708304197266296693519762960867035647583046495275004602971410174748522149063743881854897459587641941644529030594777228401015224451825864074386289782279503496785226073777246465435571069300446245182184766919094138730025163864703186 7

RSA Public Key (e, n) = (65537,
422712553676355411878798390198553638236603487356195751756211794316367919585103489956464085330259833137291140919453550689497273802216229158914797353786561555013923629763534614443739308225907911110852908207962253161953644709469980613985223184100746112863789999417274339288905236931355954698221747038250294310275564815345236550534128141819530053036870866511812043168363237013231775575698673225426086468182549050524326446688276019647736307598234680534345462590275219051629711251090566028910638366942273251339568859429319432489989304883955067942406733526657996686286011458219620143572272751053275215084863221348633843949483611065077411324551257487864300809199032696472528276198739442113667079805622171591194522122048204394133870187680060389650158419729345411080534112912640663745631721111205118276099185196802550807375243334574600878115533391008811948091608129231779445383345287296464986312249695665131725

77791133448203900526047483405919900284775721813452033616829108622556784271470658531480972801105408194125708304197266296693519762960867035647583046495275004602971410174748522149063743881854897459587641941644529030594777228401015224451825864074386289782279503496785226073777246465435571069300446245182184766919094138730025163864703186 7)

RSA Private Key (d, n) = (15770854097594589584278345726451981336532328103737342762395493511801748633253529201711252194043339151990334752310235085385198958966673816568695103677976130402132301862075140543149472550516005210143071026432221572642825528771915858206131943901074726041217219701469055449521677751817422903590673874044063576575900384973383552645238517472007155235675312506035241569662403692891847574044292932289993805382503786920930628420548754072366450183994440047218030892159878207777893510231495842002066037063194109081135538980890625820243112278129278676564058534049948390949203146565951954057228193289292510008148054725110160853270390772397594354801090717084604139630079094400726353127507739400156368385447767544500151880787739895328723993698877807756658874139613731792450660804232041969092278566120586104352517674852290464142082146773272184700317802106809195088699706923628658117217869410102201891902526894356787957478161010653286180303444271751323405573327755723735241749174990503826274993712992734727093523965138345661672180786152675019610570333617484573642442793947827246161022929625601323309806504503124514664950454460456549262115510983355675844123575706755006270263885770161780768757147895497181667524073603534015946937887893791644218915121,422712553676355411878798390198553638236603487356195751756211794316367919585103489956464085330259833137291140919453550689497273802216229158914797353786561555013923629763534614443739308225907911110852908207962253161953644709469980613985223184100746112863789999417274339288905236931355954698221747038250294310275564815345236550534128141819530053036870866511812043168363237013231775575698673225426086468182549050524326446688276019647736307598234680534345462590275219051629711251090566028910638366942273251339568859429319432489989304883955067942406733526657996686286011458219620143572272751053275215084863

221348633843949483611065077411324551257487864300809199032696472528276198739
442113667079805622171591194522122048204394133870187680060389650158419729345
411080534112912640663745631721111205118276099185196802550807375243334574600
878115533391008811948091608129231779445383345287296464986312249695665131725
777911334482039005260474834059199002847757218134520336168291086225567842714
706585314809728011054081941257083041972662966935197629608670356475830464952
750046029714101747485221490637438818548974595876419416445290305947772284010
152244518258640743862897822795034967852260737772464654355710693004462451821
847669190941387300251638647031867)

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

The result is 5.

1. Add a value to your sum.
2. Subtract a value from your sum.
4. Exit client

2

Enter value to subtract:

3

The result is 2.

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client

3

The result is 2.

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.

(Client 2 Console Output)

PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute

d Systems for ISM\Projects\Project2\Project2Task5\src> java .\SigningClient

TCP.java

The client is running.

Please enter server port: 6789

e = 65537

d = 590628297439620755773002178924694956046215792688968071777384174113987960482935016663876814404615039033230404332803636959302482354281513357621025273369286286497834932055721431562388087824520608120525485401612265347107252860208880632931281002622748437297211233086080544459456356705249616073733700823525660480318129367646756041791671651870025029457321843523943208127903109288894160352215637304955077498013031067838984125602706681886344773189058868840980872223951173787555103522329993105204979726425304584172684229154779732423121630441270363646737295903593582160881498111554348716880552302228594659073816999183426264554317066606343017626677023170294006291920055519085596620169352959960654070097582653331221110518516355869143058311682284577742166608114996711426956631790663965354717975841079082071702001575671120010114310493536464154917525507191134975171431425017018345728081096077966139367281755131 9

589138050736662685045055778676578433343693949907051492709573777812889248190
620620977427588889470091201279793950532448758279813721648524620800763322145
607805883909014330796719557042854621107971143174453794508688333505441604216
417468166920592618075881436545728664469187318704925626343112975819820902434
965976419534459069577603867361954037 77

n = 707021383964718811118127489592089817607964572321489378974107302895236 8
847477553917421731348293135057558416930072139978702748371822267030926866576
558194439131695880752198402765453391559829693704682340676694210388334435601
428273071169799328389874893025342904322123632030802840514610934479547067573
403850590140025068380117164227168259123366302030310605256635615187966388922
602881673669021009361956768098794479632673979440276281838749429641355182123
042186641145080747518630601947640234919233352146331406906722657690895886484
221181101452023055146529948424383099670799068776382215214839036354050450824
088687198707570523142251383603299553441574515035247223480904527637854989602
927400873378086599945984715377196047805208616845834332686195683873414296447
003750919867181361042754194954316424756028618743339324086073499653232809971
768691584954081636494710477418409923351695304010770152969747954449362311768
623267004930061547361249278936858168249264454131806792562967177471280926575
902115684888968423703694727085339170491132480364766012923404756386212458646
887889685282813771841254209662280993323474391049414317981834721771311284133
413507622762776871525012059689758586250671136604629691311404561559005996093
362930478295829556502554938536223940 29

RSA Public Key (e, n) = (65537,

707021383964718811118127489592089817607964572321489378974107302895236884747
755391742173134829313505755841693007213997870274837182226703092686657655819
443913169588075219840276545339155982969370468234067669421038833443560142827
307116979932838987489302534290432212363203080284051461093447954706757340385
059014002506838011716422716825912336630203031060525663561518796638892260288
167366902100936195676809879447963267397944027628183874942964135518212304218
664114508074751863060194764023491923335214633140690672265769089588648422118

1101452023055146529948424383099670799068776382215214839036354050450824088687198707570523142251383603299553441574515035247223480904527637854989602927400873378086599945984715377196047805208616845834332686195683873414296447003750919867181361042754194954316424756028618743339324086073499653232809971768691584954081636494710477418409923351695304010770152969747954449362311768623267004930061547361249278936858168249264454131806792562967177471280926575902115684888968423703694727085339170491132480364766012923404756386212458646887889685282813771841254209662280993323474391049414317981834721771311284133413507622762776871525012059689758586250671136604629691311404561559005996093362930478295829556502554938536223940291)

RSA Private Key (d, n) = (590628297439620755773002178924694956046215792688968071777384174113987960482935016663876814404615039033230404332803636959302482354281513357621025273369286286497834932055721431562388087824520608120525485401612265347107252860208880632931281002622748437297211233086080544459456356705249616073733700823525660480318129367646756041791671651870025029457321843523943208127903109288894160352215637304955077498013031067838984125602706681886344773189058868840980872223951173787555103522329993105204979726425304584172684229154779732423121630441270363646737295903593582160881498111554348716880552302228594659073816999183426264554317066606343017626677023170294006291920055519085596620169352959960654070097582653331221110518516355869143058311682284577742166608114996711426956631790663965354717975841079082071702001575671120010114310493536464154917525507191134975171431425017018345728081096077966139367281755131958913805073666268504505577867657843334369394990705149270957377781288924819062062097742758888947009120127979395053244875827981372164852462080076332214560780588390901433079671955704285462110797114317445379450868833350544160421641746816692059261807588143654572866446918731870492562634311297581982090243496597641953445906957760386736195403777,707021383964718811118127489592089817607964572321489378974107302895236884747755391742173134829313505755841693007213997870274837182226703092686657655819443913169588075219840276545339155982969370468234067669421038833443560142827

307116979932838987489302534290432212363203080284051461093447954706757340385
059014002506838011716422716825912336630203031060525663561518796638892260288
167366902100936195676809879447963267397944027628183874942964135518212304218
664114508074751863060194764023491923335214633140690672265769089588648422118
110145202305514652994842438309967079906877638221521483903635405045082408868
719870757052314225138360329955344157451503524722348090452763785498960292740
087337808659994598471537719604780520861684583433268619568387341429644700375
091986718136104275419495431642475602861874333932408607349965323280997176869
158495408163649471047741840992335169530401077015296974795444936231176862326
700493006154736124927893685816824926445413180679256296717747128092657590211
568488896842370369472708533917049113248036476601292340475638621245864688788
968528281377184125420966228099332347439104941431798183472177131128413341350
762276277687152501205968975858625067113660462969131140456155900599609336293
047829582955650255493853622394029)

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

The result is 10.

1. Add a value to your sum.

2. Subtract a value from your sum.

4. Exit client

2

Enter value to subtract:

20

The result is -10.

1. Add a value to your sum.

2. Subtract a value from your sum.

3. Get your sum.

4. Exit client

3

The result is -10.

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.

(Client 3 Console Output)

PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute

d Systems for ISM\Projects\Project2\Project2Task5\src> java .\SigningClient

TCP.java

The client is running.

Please enter server port: 6789

e = 65537

d = 2880194474325357871722872015634765484034087060318651683059189338482018

066706697098208155605997863453753769779147145447433166078648594175594691954

712197794162338739441219276205335958141767632485492897008570644363366015563

318723870572822404666300356970516042944437427307433955277652794198794598082

9428004104399102507043688201565419753162134292737370407656705607949311424 75

7431940349831101374106894126920397494366653406597502922838547244936485384 98

4895600039859329607028423618758123642976211810060215756627275018239716548 17

5759530295855734528817763569414142039874792577521465792484224118374773705638393332519376243947093593849799553028593257747049743782840178207925447314439263039152890272186149025536945006609645500841948138501338417298667568236478884713261326467319129998510905981767462307441840516714164645951015058155237499237592499611404485798887348076750233250019960940537110004981404923363244480136576107464052843830221370588087083767554735777317585486249165222346324328082672280408325180493401361580034130711268233759432599546040864076290732458123427676672224301007166463682648893783048031308146952520885607226354963163025400687752489009857990432220707277925842834663665697862828496514640797422825750404745691952660582374094588 1

n = 3825219982650285308618770787675913458581079797189305624623071610183115511647485261728770193131803533592607506504407068220877179434866381995487509341654983909885177456312319228896013633060943748262229790675927524863586956860114422764379320200529455169146636129566645188005089106138920548907744793130594384529655860665752429104674977430768036199750829665216056880342931229132689731096504054543110931779641897191884578908484069740343433695542582328365222996417664824820293030240650000384493157289175130794961542719523980500536522386818581594891439571695161972332659751860382409196365654857389746578088160810040045595511022733122154585343852864862807081653646624810973076207414870290077566110239935841328740223103364727622927081187455081661815733277151919741097336740041202785354346370536505902399569817028826718722898237797308605941859537614975382776660281725321008624640754111437706175485190659157901031434572227277004119894644570595769128504426220372362333514187686731670815722989354615999130884317461816075122166244640613938296021873142350219683150629056246200730994473280849016716134616862072350830740220267358347684387418090358226518450518197198351909675384025640370793248484472893886711741685835190110319933540603096678976002107 7

RSA Public Key (e, n) = (65537,

3825219982650285308618770787675913458581079797189305624623071610183115511647485261728770193131803533592607506504407068220877179434866381995487509

49839098851774563123192288960136330609437482622297906759275248635869568601144227643793202005294551691466361295666451880050891061389205489077447931305943845296558606657524291046749774307680361997508296652160568803429312291326897310965040545431109317796418971918845789084840697403434336955425823283652229964176648248202930302406500003844931572891751307949615427195239805005365223868185815948914395716951619723326597518603824091963656548573897465780881608100400455955110227331221545853438528648628070816536466248109730762074148702900775661102399358413287402231033647276229270811874550816618157332771519197410973367400412027853543463705365059023995698170288267187228982377973086059418595376149753827766602817253210086246407541114377061754851906591579010314345722272770041198946445705957691285044262203723623335141876867316708157229893546159991308843174618160751221662446406139382960218731423502196831506290562462007309944732808490167161346168620723508307402202673583476843874180903582265184505181971983519096753840256403707932484844728938867117411685835190110319933540603096678976002107 7)

RSA Private Key (d, n) = (288019447432535787172287201563476548403408706031865168305918933848201806670669709820815560599786345375376977914714544743316607864859417559469195471219779416233873944121927620533595814176763248549289700857064436336601556331872387057282240466630035697051604294443742730743395527765279419879459808294280041043991025070436882015654197531621342927373704076567056079493114247574319403498311013741068941269203974943666534065975029228385472449364853849848956000398593296070284236187581236429762118100602157566272750182397165481757595302958557345288177635694141420398747925775214657924842241183747737056383933325193762439470935938497995530285932577470497437828401782079254473144392630391528902721861490255369450066096455008419481385013384172986675682364788847132613264673191299985109059817674623074418405167141646459510150581552374992375924996114044857988873480767502332500199609405371100049814049233632444801365761074640528438302213705880870837675547357757758548624916522234632432808267228040832518049340136158003413071126823375943259954604086407629073245812342767667222430100716646368264889378304803130814

695252088560722635496316302540068775248900985799043222070727792584283466366
569786282849651464079742282575040474569195266058237409458 81,
382521998265028530861877078767591345858107979718930562462307161018311551164
748526172877019313180353359260750650440706822087717943486638199548750934165
498390988517745631231922889601363306094374826222979067592752486358695686011
442276437932020052945516914663612956664518800508910613892054890774479313059
438452965586066575242910467497743076803619975082966521605688034293122913268
973109650405454311093177964189719188457890848406974034343369554258232836522
299641766482482029303024065000038449315728917513079496154271952398050053652
238681858159489143957169516197233265975186038240919636565485738974657808816
081004004559551102273312215458534385286486280708165364662481097307620741487
029007756611023993584132874022310336472762292708118745508166181573327715191
974109733674004120278535434637053650590239956981702882671872289823779730860
594185953761497538277660281725321008624640754111437706175485190659157901 03
143457222727700411989464457059576912850442622037236233351418768673167081572
298935461599913088431746181607512216624464061393829602187314235021968315062
905624620073099447328084901671613461686207235083074022026735834768438741809
035822651845051819719835190967538402564037079324848447289388671174116858351
901103199335406030966789760021077)

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

The result is -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:

-40

The result is 30.

1. Add a value to your sum.

2. Subtract a value from your sum.

3. Get your sum.

4. Exit client

3

The result is 30.

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.

PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute

d Systems for ISM\Projects\Project2\Project2Task5\src>

Project2Task5ServerConsole

Note: I am pasting the output of my Server Console here, because taking multiple screenshots of such a huge output (displaying the interaction of 3 different clients) was not seeming the best possible representation. My comments in the output are mentioned in orange for better understanding of the grader.

(Server output for client 1)

PS F:\Shivam\CMU Material\A - CMU Study Material\Semester 3\S1 - Distribute

d Systems for ISM\Projects\Project2\Project2Task5\src> java .\VerifyingServ

erTCP.java

Server started

Visitor's Public Key Material:

e = 65537

n = 4227125536763554118787983901985536382366034873561957517562117943163679

195851034899564640853302598331372911409194535506894972738022162291589147973

537865615550139236297635346144437393082259079111108529082079622531619536447

094699806139852231841007461128637899994172743392889052369313559546982217470

382502943102755648153452365505341281418195300530368708665118120431683632370

132317755756986732254260864681825490505243264466882760196477363075982346805

343454625902752190516297112510905660289106383669422732513395688594293194324

899893048839550679424067335266579966862860114582196201435722727510532752150

848632213486338439494836110650774113245512574878643008091990326964725282761

987394421136670798056221715911945221220482043941338701876800603896501584197

293454110805341129126406637456317211112051182760991851968025508073752433345

746008781155333910088119480916081292317794453833452872964649863122496956651

317257779113344820390052604748340591990028477572181345203361682910862255678

427147065853148097280110540819412570830419726629669351976296086703564758304

649527500460297141017474852214906374388185489745958764194164452903059477722

840101522445182586407438628978227950349678522607377724646543557106930044624

51821847669190941387300251638647031867

Visitor's Public Key (e, n) = (65537,

422712553676355411878798390198553638236603487356195751756211794316367919585

103489956464085330259833137291140919453550689497273802216229158914797353786

561555013923629763534614443739308225907911110852908207962253161953644709469

980613985223184100746112863789999417274339288905236931355954698221747038250

294310275564815345236550534128141819530053036870866511812043168363237013231

775575698673225426086468182549050524326446688276019647736307598234680534345

462590275219051629711251090566028910638366942273251339568859429319432489989

304883955067942406733526657996686286011458219620143572272751053275215084863

221348633843949483611065077411324551257487864300809199032696472528276198739

442113667079805622171591194522122048204394133870187680060389650158419729345

411080534112912640663745631721111205118276099185196802550807375243334574600

878115533391008811948091608129231779445383345287296464986312249695665131725

777911334482039005260474834059199002847757218134520336168291086225567842714

706585314809728011054081941257083041972662966935197629608670356475830464952

750046029714101747485221490637438818548974595876419416445290305947772284010

152244518258640743862897822795034967852260737772464654355710693004462451821

847669190941387300251638647031867)

Signature verified!

Public Key hash Client ID verified!

Visitor ID: 1F90313CFC56A6838F6A7398F3838F6421F4A6CC

Operation Requested: 1. Add

Returning sum of 5 to client.

Visitor's Public Key Material:

e = 65537

n = 422712553676355411878798390198553638236603487356195751756211794316367 9

19585103489956464085330259833137291140919453550689497273802216229158914797353786156155013923629763534614443739308225907911110852908207962253161953644709469806139852231841007461128637899994172743392889052369313559546982217470382502943102755648153452365505341281418195300530368708665118120431683632370132317755756986732254260864681825490505243264466882760196477363075982346805343454625902752190516297112510905660289106383669422732513395688594293194324899893048839550679424067335266579966862860114582196201435722727510532752150848632213486338439494836110650774113245512574878643008091990326964725282761987394421136670798056221715911945221220482043941338701876800603896501584197293454110805341129126406637456317211112051182760991851968025508073752433345746008781155333910088119480916081292317794453833452872964649863122496956651317257779113344820390052604748340591990028477572181345203361682910862255678427147065853148097280110540819412570830419726629669351976296086703564758304649527500460297141017474852214906374388185489745958764194164452903059477722840101522445182586407438628978227950349678522607377724646543557106930044624518218476691909413873002516386470318671

Visitor's Public Key (e, n) = (65537,

422712553676355411878798390198553638236603487356195751756211794316367919585103489956464085330259833137291140919453550689497273802216229158914797353786561555013923629763534614443739308225907911110852908207962253161953644709469806139852231841007461128637899994172743392889052369313559546982217470382502943102755648153452365505341281418195300530368708665118120431683632370132317755756986732254260864681825490505243264466882760196477363075982346805343454625902752190516297112510905660289106383669422732513395688594293194324899893048839550679424067335266579966862860114582196201435722727510532752150848632213486338439494836110650774113245512574878643008091990326964725282769873944211366707980562217159119452212204820439413387018768006038965015841972934541108053411291264066374563172111120511827609918519680255080737524333457460087811553339100881194809160812923177944538334528729646498631224969566513172577791133448203900526047483405919900284775721813452033616829108622556784271470658531480972801105408194125708304197266296693519762960867035647583046495275004602971410174748522149063743881854897459587641941644529030594777228401015224451825864074386289782279503496785226073777246465435571069300446245182186476691909413873002516386470318671

Visitor's Public Key (e, n) = (65537,

422712553676355411878798390198553638236603487356195751756211794316367919585103489956464085330259833137291140919453550689497273802216229158914797353786561555013923629763534614443739308225907911110852908207962253161953644709469806139852231841007461128637899994172743392889052369313559546982217470382502943102755648153452365505341281418195300530368708665118120431683632370132317755756986732254260864681825490505243264466882760196477363075982346805343454625902752190516297112510905660289106383669422732513395688594293194324899893048839550679424067335266579966862860114582196201435722727510532752150848632213486338439494836110650774113245512574878643008091990326964725282769873944211366707980562217159119452212204820439413387018768006038965015841972934541108053411291264066374563172111120511827609918519680255080737524333457460087811553339100881194809160812923177944538334528729646498631224969566513172577791133448203900526047483405919900284775721813452033616829108622556784271

706585314809728011054081941257083041972662966935197629608670356475830464952
750046029714101747485221490637438818548974595876419416445290305947772284010
152244518258640743862897822795034967852260737772464654355710693004462451821
847669190941387300251638647031867)

Signature verified!

Public Key hash Client ID verified!

Visitor ID: 1F90313CFC56A6838F6A7398F3838F6421F4A6CC

Operation Requested: 2. Subtract

Returning sum of 2 to client.

Visitor's Public Key Material:

e = 65537

n = 422712553676355411878798390198553638236603487356195751756211794316367919585103489956464085330259833137291140919453550689497273802216229158914797353786561555013923629763534614443739308225907911110852908207962253161953644709469980613985223184100746112863789999417274339288905236931355954698221747038250294310275564815345236550534128141819530053036870866511812043168363237013231775575698673225426086468182549050524326446688276019647736307598234680534345462590275219051629711251090566028910638366942273251339568859429319432489989304883955067942406733526657996686286011458219620143572272751053275215084863221348633843949483611065077411324551257487864300809199032696472528276198739442113667079805622171591194522122048204394133870187680060389650158419729345411080534112912640663745631721111205118276099185196802550807375243334574600878115533391008811948091608129231779445383345287296464986312249695665131725777911334482039005260474834059199002847757218134520336168291086225567842714706585314809728011054081941257083041972662966935197629608670356475830464952750046029714101747485221490637438818548974595876419416445290305947722840101522445182586407438628978227950349678522607377724646543557106930044624

5182184766919094138730025163864703186 7

Visitor's Public Key (e, n) = (65537,

42271255367635541187879839019855363823660348735619575175621179431636791958 5

10348995646408533025983313729114091945355068949727380221622915891479735378 6

56155501392362976353461444373930822590791111085290820796225316195364470946 9

98061398522318410074611286378999941727433928890523693135595469822174703825 0

29431027556481534523655053412814181953005303687086651181204316836323701323 1

77557569867322542608646818254905052432644668827601964773630759823468053434 5

46259027521905162971125109056602891063836694227325133956885942931943248998 9

30488395506794240673352665799668628601145821962014357227275105327521508486 3

22134863384394948361106507741132455125748786430080919903269647252827619873 9

44211366707980562217159119452212204820439413387018768006038965015841972934 5

41108053411291264066374563172111120511827609918519680255080737524333457460 0

87811553339100881194809160812923177944538334528729646498631224969566513172 5

77791133448203900526047483405919900284775721813452033616829108622556784271 4

70658531480972801105408194125708304197266296693519762960867035647583046495 2

75004602971410174748522149063743881854897459587641941644529030594777228401 0

15224451825864074386289782279503496785226073777246465435571069300446245182 1

84766919094138730025163864703186 7)

Signature verified!

Public Key hash Client ID verified!

Visitor ID: 1F90313CFC56A6838F6A7398F3838F6421F4A6CC

Operation Requested: 3. Get

Returning sum of 2 to client.

(Server output for client 2)

Visitor's Public Key Material:

e = 65537

n = 70702138396471881111812748959208981760796457232148937897410730289523688474775539174217313482931350575584169300721399787027483718222670309268665765581944391316958807521984027654533915598296937046823406766942103883344356014282730711697993283898748930253429043221236320308028405146109344795470675734038505901400250683801171642271682591233663020303106052566356151879663889226028816736690210093619567680987944796326739794402762818387494296413551821230421866411450807475186306019476402349192333521463314069067226576908958864842211811014520230551465299484243830996707990687763822152148390363540504508240886871987075705231422513836032995534415745150352472234809045276378549896029274008733780865999459847153771960478052086168458343326861956838734142964470037509198671813610427541949543164247560286187433393240860734996532328099717686915849540816364947104774184099233516953040107701529697479544493623117686232670049300615473612492789368581682492644541318067925629671774712809265759021156848889684237036947270853391704911324803647660129234047563862124586468878896852828137718412542096622809933234743910494143179818347217713112841334135076227627768715250120596897585862506711366046296913114045615590059960933629304782958295565025549385362239402 9

087337808659994598471537719604780520861684583433268619568387341429644700375
091986718136104275419495431642475602861874333932408607349965323280997176869
158495408163649471047741840992335169530401077015296974795444936231176862326
700493006154736124927893685816824926445413180679256296717747128092657590211
568488896842370369472708533917049113248036476601292340475638621245864688788
968528281377184125420966228099332347439104941431798183472177131128413341350
762276277687152501205968975858625067113660462969131140456155900599609336293
047829582955650255493853622394029)

Signature verified!

Public Key hash Client ID verified!

Visitor ID: CAC6564988E747FC0807D65E6E129B50C5D6EFAA
Operation Requested: 1. Add
Returning sum of 10 to client.

Visitor's Public Key Material:
e = 65537
n = 70702138396471881111812748959208981760796457232148937897410730289523688
847477553917421731348293135057558416930072139978702748371822267030926866576
558194439131695880752198402765453391559829693704682340676694210388334435601
428273071169799328389874893025342904322123632030802840514610934479547067573
403850590140025068380117164227168259123366302030310605256635615187966388922
602881673669021009361956768098794479632673979440276281838749429641355182123
042186641145080747518630601947640234919233352146331406906722657690895886484
221181101452023055146529948424383099670799068776382215214839036354050450824
088687198707570523142251383603299553441574515035247223480904527637854989602
927400873378086599945984715377196047805208616845834332686195683873414296447
003750919867181361042754194954316424756028618743339324086073499653232809971
768691584954081636494710477418409923351695304010770152969747954449362311768

623267004930061547361249278936858168249264454131806792562967177471280926575
902115684888968423703694727085339170491132480364766012923404756386212458646
887889685282813771841254209662280993323474391049414317981834721771311284133
413507622762776871525012059689758586250671136604629691311404561559005996093
362930478295829556502554938536223940 29

Visitor's Public Key (e, n) = (65537,
707021383964718811118127489592089817607964572321489378974107302895236884747
755391742173134829313505755841693007213997870274837182226703092686657655819
443913169588075219840276545339155982969370468234067669421038833443560142827
307116979932838987489302534290432212363203080284051461093447954706757340385
059014002506838011716422716825912336630203031060525663561518796638892260288
167366902100936195676809879447963267397944027628183874942964135518212304218
664114508074751863060194764023491923335214633140690672265769089588648422118
110145202305514652994842438309967079906877638221521483903635405045082408868
719870757052314225138360329955344157451503524722348090452763785498960292740
087337808659994598471537719604780520861684583433268619568387341429644700375
091986718136104275419495431642475602861874333932408607349965323280997176869
158495408163649471047741840992335169530401077015296974795444936231176862326
700493006154736124927893685816824926445413180679256296717747128092657590211
568488896842370369472708533917049113248036476601292340475638621245864688788
968528281377184125420966228099332347439104941431798183472177131128413341350
762276277687152501205968975858625067113660462969131140456155900599609336293
047829582955650255493853622394029)

Signature verified!

Public Key hash Client ID verified!

Visitor ID: CAC6564988E747FC0807D65E6E129B50C5D6EFAA

Operation Requested: 2. Subtract

Returning sum of -10 to client.

Visitor's Public Key Material:

e = 65537

n = 70702138396471881111812748959208981760796457232148937897410730289523688474775539174217313482931350575584169300721399787027483718222670309268665765581944391316958807521984027654533915598296937046823406766942103883344356014282730711697993283898748930253429043221236320308028405146109344795470675734038505901400250683801171642271682591233663020303106052566356151879663889226028816736690210093619567680987944796326739794402762818387494296413551821230421866411450807475186306019476402349192333521463314069067226576908958864842211811014520230551465299484243830996707990687763822152148390363540504508240886871987075705231422513836032995534415745150352472234809045276378549896029274008733780865999459847153771960478052086168458343326861956838734142964470037509198671813610427541949543164247560286187433393240860734996532328099717686915849540816364947104774184099233516953040107701529697479544493623117686232670049300615473612492789368581682492644541318067925629671774712809265759021156848889684237036947270853391704911324803647660129234047563862124586468878896852828137718412542096622809933234743910494143179818347217713112841334135076227627768715250120596897585862506711366046296913114045615590059960933629304782958295565025549385362239402 9

Visitor's Public Key (e, n) = (65537,

707021383964718811118127489592089817607964572321489378974107302895236884747755391742173134829313505755841693007213997870274837182226703092686657655819443913169588075219840276545339155982969370468234067669421038833443560142827307116979932838987489302534290432212363203080284051461093447954706757340385059014002506838011716422716825912336630203031060525663561518796638892260288167366902100936195676809879447963267397944027628183874942964135518212304218664114508074751863060194764023491923335214633140690672265769089588648422118110145202305514652994842438309967079906877638221521483903635405045082408868

719870757052314225138360329955344157451503524722348090452763785498960292740
087337808659994598471537719604780520861684583433268619568387341429644700375
091986718136104275419495431642475602861874333932408607349965323280997176869
158495408163649471047741840992335169530401077015296974795444936231176862326
700493006154736124927893685816824926445413180679256296717747128092657590211
568488896842370369472708533917049113248036476601292340475638621245864688788
968528281377184125420966228099332347439104941431798183472177131128413341350
762276277687152501205968975858625067113660462969131140456155900599609336293
047829582955650255493853622394029)

Signature verified!

Public Key hash Client ID verified!

Visitor ID: CAC6564988E747FC0807D65E6E129B50C5D6EFAA

Operation Requested: 3. Get

Returning sum of -10 to client.

(Server output for client 3)

Visitor's Public Key Material:

e = 65537

n = 382521998265028530861877078767591345858107979718930562462307161018311 5
511647485261728770193131803533592607506504407068220877179434866381995487509
341654983909885177456312319228896013633060943748262229790675927524863586956
860114422764379320200529455169146636129566645188005089106138920548907744793
130594384529655860665752429104674977430768036199750829665216056880342931229
132689731096504054543110931779641897191884578908484069740343433695542582328
365222996417664824820293030240650000384493157289175130794961542719523980500
536522386818581594891439571695161972332659751860382409196365654857389746578
088160810040045595511022733122154585343852864862807081653646624810973076207
414870290077566110239935841328740223103364727622927081187455081661815733277

151919741097336740041202785354346370536505902399569817028826718722898237797
308605941859537614975382776660281725321008624640754111437706175485190659157
901031434572227277004119894644570595769128504426220372362333514187686731670
815722989354615999130884317461816075122166244640613938296021873142350219683
150629056246200730994473280849016716134616862072350830740220267358347684387
418090358226518450518197198351909675384025640370793248484472893886711741168
583519011031993354060309667897600210 77

Visitor's Public Key (e, n) = (65537,
382521998265028530861877078767591345858107979718930562462307161018311551164
748526172877019313180353359260750650440706822087717943486638199548750934165
498390988517745631231922889601363306094374826222979067592752486358695686011
442276437932020052945516914663612956664518800508910613892054890774479313059
438452965586066575242910467497743076803619975082966521605688034293122913268
973109650405454311093177964189719188457890848406974034343369554258232836522
299641766482482029303024065000038449315728917513079496154271952398050053652
238681858159489143957169516197233265975186038240919636565485738974657808816
081004004559551102273312215458534385286486280708165364662481097307620741487
029007756611023993584132874022310336472762292708118745508166181573327715191
974109733674004120278535434637053650590239956981702882671872289823779730860
594185953761497538277660281725321008624640754111437706175485190659157901 03
143457222727700411989464457059576912850442622037236233514187686731670815 72
298935461599913088431746181607512216624464061393829602187314235021968315062
905624620073099447328084901671613461686207235083074022026735834768438741809
035822651845051819719835190967538402564037079324848447289388671174116858351
901103199335406030966789760021077)

Signature verified!

Public Key hash Client ID verified!

Visitor ID: AD3A503261AFAB729ABF5890B1AF7FD12354F7E4

Operation Requested: 1. Add

Returning sum of -10 to client.

Visitor's Public Key Material:

e = 65537

n = 382521998265028530861877078767591345858107979718930562462307161018311551164748526172877019313180353359260750650440706822087717943486638199548750934165498390988517745631231922889601363306094374826222979067592752486358695686011442276437932020052945516914663612956664518800508910613892054890774479313059438452965586066575242910467497743076803619975082966521605688034293122913268973109650405454311093177964189719188457890848406974034343369554258232836522

29964176648248202930302406500003844931572891751307949615427195239805005365223868185815948914395716951619723326597518603824091963656548573897465780881608100400455955110227331221545853438528648628070816536466248109730762074148702900775661102399358413287402231033647276229270811874550816618157332771519197410973367400412027853543463705365059023995698170288267187228982377973086059418595376149753827766602817253210086246407541114377061754851906591579010314345722272770041198946445705957691285044262203723623335141876867316708157229893546159991308843174618160751221662446406139382960218731423502196831506290562462007309944732808490167161346168620723508307402202673583476843874180903582265184505181971983519096753840256403707932484844728938867117411685835190110319933540603096678976002107 7)

Signature verified!

Public Key hash Client ID verified!

Visitor ID: AD3A503261AFAB729ABF5890B1AF7FD12354F7E4

Operation Requested: 2. Subtract

Returning sum of 30 to client.

Visitor's Public Key Material:

e = 65537

n = 38252199826502853086187707876759134585810797971893056246230716101831155116474852617287701931318035335926075065044070682208771794348663819954875093416549839098851774563123192288960136330609437482622297906759275248635869568601144227643793202005294551691466361295666451880050891061389205489077447931305943845296558606657524291046749774307680361997508296652160568803429312291326897310965040545431109317796418971918845789084840697403434336955425823283652229964176648248202930302406500003844931572891751307949615427195239805005365223868185815948914395716951619723326597518603824091963656548573897465780881608100400455955110227331221545853438528648628070816536466248109730762077

414870290077566110239935841328740223103364727622927081187455081661815733277
151919741097336740041202785354346370536505902399569817028826718722898237797
308605941859537614975382776660281725321008624640754111437706175485190659157
901031434572227277004119894644570595769128504426220372362333514187686731670
815722989354615999130884317461816075122166244640613938296021873142350219683
150629056246200730994473280849016716134616862072350830740220267358347684387
418090358226518450518197198351909675384025640370793248484472893886711741168
583519011031993354060309667897600210 77

Visitor's Public Key (e, n) = (65537,
382521998265028530861877078767591345858107979718930562462307161018311551164
748526172877019313180353359260750650440706822087717943486638199548750934165
498390988517745631231922889601363306094374826222979067592752486358695686011
442276437932020052945516914663612956664518800508910613892054890774479313059
438452965586066575242910467497743076803619975082966521605688034293122913268
973109650405454311093177964189719188457890848406974034343369554258232836522
299641766482482029303024065000038449315728917513079496154271952398050053652
238681858159489143957169516197233265975186038240919636565485738974657808816
081004004559551102273312215458534385286486280708165364662481097307620741487
029007756611023993584132874022310336472762292708118745508166181573327715191
974109733674004120278535434637053650590239956981702882671872289823779730860
594185953761497538277660281725321008624640754111437706175485190659157901 03
143457222727700411989464457059576912850442622037236233351418768673167081572
298935461599913088431746181607512216624464061393829602187314235021968315062
905624620073099447328084901671613461686207235083074022026735834768438741809
035822651845051819719835190967538402564037079324848447289388671174116858351
901103199335406030966789760021077)

Signature verified!

Public Key hash Client ID verified!

Visitor ID: AD3A503261AFAB729ABF5890B1AF7FD12354F7E4

Operation Requested: 3. Get

Returning sum of 30 to client.

Note: In the above outputs wherever the Task (Task 3, Task 4, Task 5) requested to show the "operation requested" in the Server console, I believe (and have assumed) that the prompt is asking for what the operation is about (e.g., addition, subtraction, get) and not the details of the operations (e.g., Adding 3 to 0, or Subtracting 5 from 6, etc. as given in Task 2's output).