PROJECT 2

Name: Shivani Poovaiah Ajjikutira

Email ID: sajjikut@andrew.cmu.edu

1. Project2Task1Client

```
/*
* @author: Shivani Poovaiah Ajjikutira
* Last Modified: 8th October 2021
*
* The following code is written for UDP Client. The code creates a
* socket and established connection with the server. It accepts
* string input from user, converts the string into bytes. The bytes
* of data are then added to a DatagramPacket and sent to the server. The
* server sends a response as a DatagramPacket which has bytes of reply.
* The bytes of reply are copied into a byte array of correct size based
* on the reply data. The correct sized byte array is then converted into a
* string and displayed to the user. This process continues as long as
* the user does not enter the message "halt!". When the user, enters
* "halt!" the socket is closed and connection is terminated.
* */

import java.net.*;
import java.io.*;

public class EchoClientUDP{
    public static void main(String [] args){
        // Project 2 Code from EchoClientUDP.java
        System.out.println("The client is running.");
        // Client DatagramSocket to send data to server declared
        DatagramSocket aSocket = null;
        try {
            // gets IP address of local host
            InetAddress aHost = InetAddress.getByName("localhost");
            // server port number
            int serverPort = 6789;
            // Client DatagramSocket to send data to server declared
            aSocket = new DatagramSocket();
            String nextLine;
            String checkString;
            // reads user input from console
            BufferedReader typed = new BufferedReader(new
InputStreamReader(System.in));
            while (typed!=null && (nextLine = typed.readLine()) != null) {
                // converts user input string to byte array
                byte [] m = nextLine.getBytes();
                // DatagramPacket initialized with message bytes, length of
array, host IP address and server port number
                DatagramPacket request = new DatagramPacket(m,  m.length,
aHost, serverPort);
                // data packet sent through DataSocket to server
                aSocket.send(request);
                // default buffer array to initialize DatagramPacket storing
server response data
```

```
                byte[] buffer = new byte[1000];
                // user input string
                checkString = new String(m);
                /*
                * If user enters "halt!" close the Client DatagramSocket else
receive
                * reply bytes from server. Check the length of the reply and
create
                * a new byte array of required size. Convert this new byte
array into
                * a string and display to the user.
                * */
                if(!checkString.equals("halt!")) {
                    // DatagramPacket initialized with buffer array of size
1000 bytes
                    DatagramPacket reply = new DatagramPacket(buffer,
buffer.length);
                    // data packet received from server
                    aSocket.receive(reply);
                    // byte array of correct size based on reply
                    byte[] replyBytes = new byte[reply.getLength()];
                    // copy data from DatagramPacket to correct sized array
                    for (int i = 0; i < reply.getLength(); i++) {
                        replyBytes[i] = reply.getData()[i];
                    }
                    // convert reply data in bytes to string
                    String replyString = new String(replyBytes);
                    // print reply on console
                    System.out.println("\rReply: " + replyString);
                } else {
                    System.out.println("Client side quitting.");
                    typed = null;
                    // close Client DatagramSocket and end connection with
server
                    aSocket.close();
                }
            }

        }catch (SocketException e) {
            // to catch errors when errors occur with the network
            System.out.println("Socket: " + e.getMessage());
        }catch (IOException e){
            // to catch errors when there is an input-output exception
            System.out.println("IO: " + e.getMessage());
        }finally {
            // closing the socket if not closed earlier
            if(aSocket != null) aSocket.close();
        }
    }
}
```

2. Project2Task1Server

```
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;

/*
 * @author: Shivani Poovaiah Ajjikutira
 * Last Modified: 8th October 2021
 *
 * The following code is written for UDP Server. The code creates a
 * socket and with port number 6789. It receives a DatagramPacket from
 * the client which established a connection with this server.
 * The bytes of request are copied into a byte array of correct size based
 * on the request data. The correct sized byte array is then converted into a
 * string and displayed to the user as "Echoing". This process continues as
long as
 * the user does not send message "halt!". When the user, sends "halt!" the
socket
 * is closed and connection is terminated.
 * */
public class EchoServerUDP{
    public static void main(String[] args){
        // Project 2 Code from EchoServerUDP.java
        System.out.println("The server is running.");
        // Server DatagramSocket to receive data from the client declared
        DatagramSocket aSocket = null;
        // default buffer array to initialize DatagramPacket storing client
request data
        byte[] buffer = new byte[1000];
        try{
            // Server socket initialized with server port number
            aSocket = new DatagramSocket(6789);
            // DatagramPacket initialized with default buffer array and size
of buffer array to receive
            // data from client socket
            DatagramPacket request = new DatagramPacket(buffer,
buffer.length);
            // to ensure the client is always running
            while(true){
                // data packet received from client socket connected to
server port 6789
                aSocket.receive(request);
                // prints client's port number
                System.out.println("Sending data to port number: " +
request.getPort());
                // DatagramPacket initialized with request data, length of
request data, client IP address and client port number
                DatagramPacket reply = new DatagramPacket(request.getData(),
                        request.getLength(), request.getAddress(),
request.getPort());
                // byte array of correct size based on request
                byte[] requestBytes = new byte[request.getLength()];
```

```java
                // copy data from request DatagramPacket to correct sized
array
                System.arraycopy(request.getData(), 0, requestBytes, 0,
request.getLength());
                // convert request data in bytes to string
                String requestString = new String(requestBytes);
                // print request string
                System.out.println("Echoing: "+requestString);
                // reply DatagramPacket sent through DataSocket to client
                aSocket.send(reply);
                // if request string is "halt!" close the connection.
                if(requestString.equals("halt!")) {
                    System.out.println("Server side quitting.");
                    // terminate socket
                    aSocket.close();
                }
            }
        }catch (SocketException e) {
            // to catch errors when errors occur with the network
            System.out.println("Socket: " + e.getMessage());
        }catch (IOException e){
            // to catch errors when there is an input-output exception
            System.out.println("IO: " + e.getMessage());
        }finally {
            // closing the socket if not closed earlier
            if(aSocket != null) aSocket.close();
        }
    }
}
```

3. Project2Task1ClientScreen

```
EchoServerUDP ×   EchoClientUDP ×
"C:\Program Files\Eclipse Foundation\jdk-16.0.2.7-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\Int
The client is running.
Submit to Canvas a single PDF file named Your_Last_Name_First_Name_Project2.pdf.
The single PDF will contain your responses to the questions marked with a checkered flag.
It is important that you clearly label each answer with the labels provided below.
It is also important to be prepared to demonstrate your working code if we need to verify your submission.
Be sure to provide your name and email address at the top of the PDF submission.
Reply: Submit to Canvas a single PDF file named Your_Last_Name_First_Name_Project2.pdf.
Reply: The single PDF will contain your responses to the questions marked with a checkered flag.
Reply: It is important that you clearly label each answer with the labels provided below.
Reply: It is also important to be prepared to demonstrate your working code if we need to verify your submission.
Reply: Be sure to provide your name and email address at the top of the PDF submission.
halt!
Client side quitting.

Process finished with exit code 0
```

4. Project2Task1ServerScreen

```
EchoServerUDP    EchoClientUDP
"C:\Program Files\Eclipse Foundation\jdk-16.0.2.7-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\Intelli
The server is running.
Sending data to port number: 62343
Echoing: Submit to Canvas a single PDF file named Your_Last_Name_First_Name_Project2.pdf.
Sending data to port number: 62343
Echoing: The single PDF will contain your responses to the questions marked with a checkered flag.
Sending data to port number: 62343
Echoing: It is important that you clearly label each answer with the labels provided below.
Sending data to port number: 62343
Echoing: It is also important to be prepared to demonstrate your working code if we need to verify your submission.
Sending data to port number: 62343
Echoing: Be sure to provide your name and email address at the top of the PDF submission.
Sending data to port number: 62343
Echoing: halt!
Server side quitting.
Socket: Socket closed

Process finished with exit code 0
```

5. Project2Task2Client

```java
/*
 * @author: Shivani Poovaiah Ajjikutira
 * Last Modified: 8th October 2021
 *
 * The following code is the client side for a program that adds numbers.
 * The code takes integer values from the user and output is the sum of
 * the integers. The client socket is initialized and forms a connection
 * with the server socket having port number 6789.The client passes the
 * integers to the server through DatagramPacket.The addition logic happens
 * in the server. The program runs till the user enters "halt!". When the
 * user enters "halt!" the connection is terminated, however, the server
 * continues to be on.
 * */

import java.net.*;
import java.io.*;

public class AddingClientUDP{
    public static void main(String [] args){
        // Project 2 Code from EchoClientUDP.java
        System.out.println("The client is running.");
        String nextLine;
        try{
            // reads user input from console
            BufferedReader typed = new BufferedReader(new
InputStreamReader(System.in));
            // keep looping as long as user does not enter "halt!"
            while ((nextLine = typed.readLine()) != null &&
```

```
!nextLine.equals("halt!")) {
                try{
                    /*
                    * The user input string is converted into an integer
using parseInt
                    * and passed as a parameter to add method. The add method
returns
                    * the sum which is stored in newSum.
                    * */
                    int newSum = add(Integer.parseInt(nextLine));
                    System.out.println("The server returned " + newSum+".");
                } catch (NumberFormatException e) {
                    // shows exception message if user enters non-integer
values
                    System.out.println("Incorrect input. Enter integers
only");
                }
            }
            System.out.println("Client side quitting.");
        }catch (IOException e){
            // to catch errors when there is an input-output exception
            System.out.println("IO: " + e.getMessage());
        }
    }

    /*
    * This method is used to create connection with the server and
    * pass the data to the server to perform the addition. The client
    * Socket is initialized. The DatagramPacket used for data transmission
    * is initialized with the user input in bytes, the length of the byte
    * array, the host IP address and the server port number. The reply
    * sent by the server is added into an array of correct size based
    * on the reply string and returned to the calling method, i.e., the
    * main method
    * */
    public static int add(int i) throws IOException {
        DatagramSocket aSocket = null;
        try {
            // Project 2 Code from EchoClientUDP.java

            // gets IP address of local host
            InetAddress aHost = InetAddress.getByName("localhost");
            // server port number
            int serverPort = 6789;
            // 4-byte byte array
            byte [] message = new byte[4];
            // converting integer into 4-byte byte array
            for(int j=0;j<message.length;j++) {
                if(j < Integer.toString(i).getBytes().length) {
                    message[j] = Integer.toString(i).getBytes()[j];
                }
                else message[j] = 0;
            }
            // client socket initialized
            aSocket = new DatagramSocket();
            // client DatagramPacket initialized
            DatagramPacket request = new DatagramPacket(message,
```

```
message.length, aHost, serverPort);
            // request sent to server through client socket
            aSocket.send(request);
            // default byte array to store server response data, initialized
DatagramPacket
            byte[] buffer = new byte[1000];
            // DatagramPacket to store server reply
            DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
            // reply from server received at client socket
            aSocket.receive(reply);
            // byte array of correct size based on reply string length
            byte[] replyBytes = new byte[reply.getLength()];
            // copy contents of reply DatagramPacket to replyBytes - syntax:
IntelliJ suggestion
            System.arraycopy(reply.getData(), 0, replyBytes, 0,
reply.getLength());
            // convert reply bytes to string
            String replyString = new String(replyBytes);
            // return reply string to calling method
            return Integer.parseInt(replyString);
        } catch (SocketException e) {
            // to catch errors when errors occur with the network
            System.out.println("Socket: " + e.getMessage());
        } finally {
                // closing the socket if not closed earlier
            if(aSocket!=null) aSocket.close();
        }
        return -1;
    }
}
```

6. Project2Task2Server

```
/*
 * @author: Shivani Poovaiah Ajjikutira
 * Last Modified: 8th October 2021
 *
 * The following code is the server side for a program that adds numbers.
 * The server socket is initialized with port number 6789 and forms a
 * connection with any client socket trying to connect to port number 6789.
 * The client passes the integers to the server through DatagramPacket.
 * The addition logic happens in the server in the add method. The program
 * on the server side keeps running always.
 * */

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;

public class AddingServletUDP {
    public static void main(String[] args){
        System.out.println("Server started");
```

```
        // to store the sum
        int sum=0;
        // Server socket
        DatagramSocket aSocket = null;
        // default byte array to store client request data in DatagramPacket
        byte[] buffer = new byte[1000];
        try{
            // Server socket initialized with port number 6789
            aSocket = new DatagramSocket(6789);
            // DatagramPacket initialized with default array and its length
            DatagramPacket request = new DatagramPacket(buffer,
buffer.length);
            // since Server is always up and running
            while(true){
                // Server receives requests sent from client socket connected
                // to port number 6789
                aSocket.receive(request);
                // DatagramPacket to send reply data back to the client
                DatagramPacket reply = new DatagramPacket(request.getData(),
                        request.getLength(), request.getAddress(),
request.getPort());
                // byte-array of correct size to store request data
                byte[] requestBytes = new byte[request.getLength()];
                // copy data from DatagramPacket to correct sized array  -
syntax: IntelliJ suggestion
                System.arraycopy(request.getData(), 0, requestBytes, 0,
request.getLength());
                // convert request bytes to string
                String requestString = new String(requestBytes);
                /*
                * Parse request string to form an integer and send it to add
                * method along with the sum to perform addition. The result
returned
                * is stored in the sum variable. */
                sum = add(Integer.parseInt(requestString.trim()), sum);
                // convert sum to byte array
                byte [] replyBytes = Integer.toString(sum).getBytes();
                // load the response DatagramPacket with the response byte
array
                reply.setData(replyBytes);
                System.out.printf("Returning sum of %d to client\n%n",sum);
                // send reply DatagramPacket back to the client socket that
sent the request
                aSocket.send(reply);
            }
        }catch (SocketException e) {
            // to catch errors when errors occur with the network
            System.out.println("Socket: " + e.getMessage());
        }catch (IOException e){
            // to catch errors when there is an input-output exception
            System.out.println("IO: " + e.getMessage());
        }
    }

    /*
    * This method performs addition of two integers and
    * returns the sum of the two integers*/
```

```
    public static int add(int i, int j) {
        System.out.printf("Adding: %d to %d%n",i,j);
        return i+j;
    }
}
```

7. Project2Task2ClientScreen

```
AddingServletUDP    AddingClientUDP
"C:\Program Files\Eclipse Foundation\jd
The client is running.
1
The server returned 1.
2
The server returned 3.
3
The server returned 6.
4
The server returned 10.
5
The server returned 15.

Process finished with exit code 130
```

```
AddingServletUDP    AddingClientUDP
"C:\Program Files\Eclipse Foundation\j
The client is running.
6
The server returned 21.
7
The server returned 28.
8
The server returned 36.
9
The server returned 45.
10
The server returned 55.
halt!
Client side quitting.

Process finished with exit code 0
```

8. Project2Task2CServerScreen

```
AddingServletUDP    AddingClientUDP
"C:\Program Files\Eclipse Foundation\jdk-16.0.
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 6 to client

Adding: 4 to 6
Returning sum of 10 to client

Adding: 5 to 10
Returning sum of 15 to client

Adding: 6 to 15
Returning sum of 21 to client

Adding: 7 to 21
Returning sum of 28 to client
```

```
Adding: 8 to 28
Returning sum of 36 to client

Adding: 9 to 36
Returning sum of 45 to client

Adding: 10 to 45
Returning sum of 55 to client
```

9. Project2Task3Client

```
/*
 * @author: Shivani Poovaiah Ajjikutira
 * Last Modified: 9th October 2021
 *
 * The following code is the client side for a program that returns a
 * number stored against a particular ID or adds/subtracts integers to
 * that integer. The sum/ difference is then stored against the ID.
 * The user id, operation and value(in case of add/subtract) are taken
 * from the user and output is the result of the corresponding operation.
 * The client socket is initialized and forms a connection with the server
 * socket having port number 6790.The client passes the data entered by the
 * user to the server through DatagramPacket via the socket connection
formed.
 * The operation logic happens in the server. The program runs till the user
 * chooses option 4, i.e, exit. When the user enters 4 the connection is
terminated,
 * however, the server continues running.
 * */

import java.net.*;
import java.io.*;

public class RemoteVariableClientUDP{
    public static void main(String [] args){
        // stores user choice from menu
        String userChoice = null;
        // to read user input from console
        BufferedReader typed = new BufferedReader(new
InputStreamReader(System.in));
        /*
         * do-while loop used since we need to run the loop at least once to
show the
         * menu and then continues looping until user choose option 4
         * */
        do {
            // displays menu options
            displayMenu();
            try {
                // checks if user input it is not null and not option 4, i.e,
Exit
                if ((userChoice = typed.readLine()) != null &&
Integer.parseInt(userChoice)!=4) {
                    // checks if user selects from the available menu options
                    if (Integer.parseInt(userChoice) > 4 ||
Integer.parseInt(userChoice) < 1) {
                        System.out.println("Please select an integer between
1-4 only");
                    } else { // if user selects a valid menu option performs
this block of code

                        // stores integer to be added/subtracted
                        int value = 0;
                        if (Integer.parseInt(userChoice) == 1) {
                            System.out.println("Enter value to add: ");
```

```
                            value = Integer.parseInt(typed.readLine());
                        } else if (Integer.parseInt(userChoice) == 2) {
                            System.out.println("Enter value to subtract: ");
                            value = Integer.parseInt(typed.readLine());
                        }
                        System.out.println("Enter your ID: ");
                        // stores ID
                        int id = Integer.parseInt(typed.readLine());
                        // checks if ID range is valid else throws exception
and displays
                        // message to user
                        if (id > 1999 || id < 1000) {
                            throw new IDOutOfRangeException("ID out of range.
Valid range: 1000-1999");
                        }
                        /*
                         * The id, operation and value entered by the user is
sent
                         * to the getResult method. The method returns the
result
                         * of the operation which is stored in result
variable
                         * */
                        int result = getResult(id,
Integer.parseInt(userChoice), value);
                        System.out.println("The result is " + result + ".");
                    }
                }
            } catch (IDOutOfRangeException e) {
                // To catch incorrect range of ID
                System.out.println(e.getMessage());
            } catch (IOException e) {
                // to catch errors when there is an input-output exception
                System.out.println("IO: " + e.getMessage());
            }
        } while (userChoice!=null && !userChoice.equals("4"));
        // exit loop when user enters 4
        System.out.println("Client side quitting.The remote variable server
is still running.");
    }

    /*
     * This method is used to create connection with the server and
     * pass the data to the server to perform the addition/subtraction
     * or return the sum stored against the ID entered by the user. The
client
     * Socket is initialized. The DatagramPacket used for data transmission
     * is initialized with the user input in bytes, the length of the byte
     * array, the host IP address and the server port number. The payload for
     * the DatagramPacket is generated based on the operation selected. The
     * request is then sent to the server to perform the logic. The reply
     * sent by the server is added into an array of correct size based
     * on the reply string and returned to the calling method, i.e., the
     * main method
     * */
    public static int getResult(int id, int userChoice, int value) throws
IOException {
```

```
        // Client socket declared
        DatagramSocket aSocket = null;
        try {
            // Client socket initialized
            aSocket = new DatagramSocket();
            // gets IP address of local host
            InetAddress aHost = InetAddress.getByName("localhost");
            // server port number
            int serverPort = 6790;
            // stores name of operation based on user choice returned
            // by getOperation method
            String operation= getOperation(userChoice);
            // stores request string to be sent to server
            String payload;
            // get operation does not require value
            if (userChoice==3) {
                payload =  id + " " + operation;
            } else {
                payload =  id + " " + operation + " " + value;
            }
            // converting payload string to bytes
            byte [] payloadBytes = payload.getBytes();
            // client DatagramPacket initialized with payloadBytes, host IP
and server port number
            DatagramPacket request = new DatagramPacket(payloadBytes,
payloadBytes.length, aHost, serverPort);;
            // request sent to server socket
            aSocket.send(request);
            // default byte array to store server response data, initialized
DatagramPacket
            byte[] buffer = new byte[1000];
            // DatagramPacket to store server reply
            DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
            // reply from server received at client socket
            aSocket.receive(reply);
            // reply store in byte array of correct size based on reply
string
            byte[] replyBytes = new byte[reply.getLength()];
            // copy contents of reply DatagramPacket to replyBytes - syntax:
IntelliJ suggestion
            System.arraycopy(reply.getData(), 0, replyBytes, 0,
reply.getLength());
            // convert reply bytes to string
            String replyString = new String(replyBytes);
            // return reply string to calling method
            return Integer.parseInt(replyString);
        } catch (SocketException e) {
            // to catch errors when errors occur with the network
            System.out.println("Socket: " + e.getMessage());
        } finally {
            // close socket connection
            if(aSocket!=null) aSocket.close();
        }
        return -1;
    }

    // returns operation name based on the user choice
```

```
    private static String getOperation(int userChoice) {
        if(userChoice==1) {
            return "add";
        } else if(userChoice==2) {
            return "subtract";
        } else {
            return "get";
        }
    }

    // displays menu options to the user
    public static void displayMenu() {
        String [] menu = {"Add a value to your sum.","Subtract a value from
your sum.",
                "Get your sum","Exit client"};
        for(int i =0; i<menu.length; i++) {
            System.out.printf("%d. %s%n",i+1,menu[i]);
        }
    }

    // New Exception created to track ID which is out of range
    public static class IDOutOfRangeException extends Exception {

        public IDOutOfRangeException(String message) {
            super(message);
        }
    }


}
```

10. Project2Task3Server

```
/*
 * @author: Shivani Poovaiah Ajjikutira
 * Last Modified: 9th October 2021
 *
 * The following code is the server side for a program that returns a
 * number stored against a particular ID or adds/subtracts integers to
 * that integer. The sum/ difference is then stored against the ID.
 * The user id, operation and value(in case of add/subtract) are sent by
 * the client and output of the corresponding operation is returned to the
 * client. The server socket is initialized and forms a connection with
 * any client socket connected to port number 6790.The server receives
 * the data from the client through DatagramPacket via the socket connection
 * formed. The performOperations method checks the operation passed and
 * does the required logic. A HashMap is used to store the integer
 * corresponding to each ID. In case of addition and subtraction the HashMap
 * values are updated, in case of get, the value for id as key is returned
 * as result to the client. The server is always running.
 * */

import java.io.IOException;
```

```java
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.SocketException;
import java.util.HashMap;

public class RemoteVariableServerUDP {
    // maps key-value pair: ID is the key, sum is the value
    static HashMap<Integer,Integer> userIdSums = new HashMap<>();
    public static void main(String[] args){
        System.out.println("Server started");
        // default byte array to store client request data, initialized
DatagramPacket
        byte[] buffer = new byte[1000];
        try{
            // Server socket initialized with port number 6790
            DatagramSocket aSocket = new DatagramSocket(6790);
            // Server DatagramPacket initialized with buffer array
            DatagramPacket request = new DatagramPacket(buffer,
buffer.length);
            // stores result of operation
            int result;
            // Since server is always running and listens to port
            while(true){
                // socket receives request from client socket
                aSocket.receive(request);
                // to store request data in correct byte size based on
request string
                byte[] requestBytes = new byte[request.getLength()];
                // copy contents of request DatagramPacket to requestBytes -
syntax: IntelliJ suggestion
                System.arraycopy(request.getData(), 0, requestBytes, 0,
request.getLength());

                // convert bytes to string
                String requestString = new String(requestBytes);
                // split data using " "
                String [] requestItems = requestString.split(" ");
                // result stores sum returned by performOperations method
                result = performOperations(requestItems);
                // convert result to byte array
                byte [] replyBytes = String.valueOf(result).getBytes();
                // initialize reply DatagramPacket with client IP address,
port number
                DatagramPacket reply = new DatagramPacket(request.getData(),
                        request.getLength(), request.getAddress(),
request.getPort());
                // load reply byte array to DatagramPacket
                reply.setData(replyBytes);
                System.out.printf("Client ID:%s,Operation:%s,Returning sum of
%d\n%n",requestItems[0],requestItems[1],result);
                // send reply DatagramPacket to client socket
                aSocket.send(reply);
            }
        }catch (SocketException e) {
            // to catch errors when errors occur with the network
            System.out.println("Socket: " + e.getMessage());
        }catch (IOException e){
```

```
            // to catch errors when there is an input-output exception
            System.out.println("IO: " + e.getMessage());
        }
    }

    /*
    * This method performs all the logical operation based on the
    * operation selected by the user as sent by the client in the
    * request. The userIdSums HashMap is used to store the key-value
    * pairs of id and sum. If a particular id does not exist and the
    * user selects "get" operation, put id and value 0 to the hashmap.
    * For any other operation, put id and the value sent from client.
    * If the id is present, then return the value stored against id in
    * the Hashmap in case the operation is "get". In case of "add"/"subtract"
    * operations add/subtract the value with the existing value in the
    * HashMap, update the value with the new sum/difference and return
    * the new sum/difference
    * */
    public static int performOperations(String [] requestItems) {
        int id = Integer.parseInt(requestItems[0]);
        String operation = requestItems[1];
        // if id not present and operation is "get"
        if(userIdSums.get(id) == null && operation.equals("get")) {
            userIdSums.put(id,0);
        } else if(userIdSums.get(id) == null && !operation.equals("get")) {
            userIdSums.put(id,Integer.parseInt(requestItems[2]));
        }
        // if id present
        else {
            if(operation.equals("add")) {
                int value = Integer.parseInt(requestItems[2]);
                int newTotal = userIdSums.get(id)+value;
                // update new total
                userIdSums.put(id,newTotal);
            } else if(operation.equals("subtract")) {
                int value = Integer.parseInt(requestItems[2]);
                int newTotal = userIdSums.get(id)-value;
                // update new total
                userIdSums.put(id,newTotal);
            }
        }
        // return latest value stores against id
        return userIdSums.get(id);
    }
}
```

11. Project2Task3ClientScreen

```
RemoteVariableServerUDP   RemoteVariableClientUDP
"C:\Program Files\Eclipse Foundation\jdk-16.0.2.7-hotspot\bin\java.exe
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:
1001
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:
2
Enter your ID:
1001
The result is 8.
```

```
RemoteVariableServerUDP   RemoteVariableClientUDP
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:
1001
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:
1002
Enter your ID:
1002
The result is 1002.
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum
4. Exit client
2
```

RemoteVariableServerUDP RemoteVariableClientUDP

```
Enter value to subtract:
20
Enter your ID:
1002
The result is 982.
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:
1002
The result is 982.
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:
1003
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:
2
Enter your ID:
1003
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:
1003
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
4
Client side quitting.The remote variable server is still running.

Process finished with exit code 0
```

RemoteVariableServerUDP RemoteVariableClientUDP

```
"C:\Program Files\Eclipse Foundation\jdk-16.0.2.7-hotspot\bin\java.e
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:
1001
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:
1002
The result is 982.
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:
1003
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
4
Client side quitting.The remote variable server is still running.

Process finished with exit code 0
```

12. Project2Task3ServerScreen

```
RemoteVariableServerUDP    RemoteVariableClientUDP
"C:\Program Files\Eclipse Foundation\jdk-16.0.2.7-hotspot\bin\java.
Server started
Client ID:1001,Operation:add,Returning sum of 10

Client ID:1001,Operation:subtract,Returning sum of 8

Client ID:1001,Operation:get,Returning sum of 8

Client ID:1002,Operation:add,Returning sum of 1002

Client ID:1002,Operation:subtract,Returning sum of 982

Client ID:1002,Operation:get,Returning sum of 982

Client ID:1003,Operation:add,Returning sum of 10

Client ID:1003,Operation:subtract,Returning sum of 8

Client ID:1003,Operation:get,Returning sum of 8
```

```
Client ID:1001,Operation:get,Returning sum of 8

Client ID:1002,Operation:get,Returning sum of 982

Client ID:1003,Operation:get,Returning sum of 8
```

13. Project2Task4Client

```
/*
 * @author: Shivani Poovaiah Ajjikutira
 * Last Modified: 9th October 2021
 *
 * This code follows Task 3 but uses TCP instead of UDP for the data
 * transmission between the client and server.
 * The following code is the client side for a program that returns a
 * number stored against a particular ID or adds/subtracts integers to
 * that integer. The sum/ difference is then stored against the ID.
 * The user id, operation and value(in case of add/subtract) are taken
 * from the user and output is the result of the corresponding operation.
 * The client socket is initialized and forms a connection with the server
 * socket having port number 7777.The client passes the data entered by the
 * user to the server via the socket connection formed. The operation logic
 * happens in the server. The program runs till the user chooses option 4,
 * i.e, exit. When the user enters 4 the connection is terminated,however,
 * the server continues running.
 * */

import java.net.*;
import java.io.*;

public class RemoteVariableClientTCP {

    public static void main(String [] args){
        // stores user choice from menu
        String userChoice = null;
        // to read user input from console
        BufferedReader typed = new BufferedReader(new
InputStreamReader(System.in));
        /*
         * do-while loop used since we need to run the loop at least once to
show the
         * menu and then continues looping until user choose option 4
         * */
        do {
            // displays menu options
            displayMenu();
            try {
                // checks if user input it is not null and not option 4, i.e,
Exit
                if ((userChoice = typed.readLine()) != null &&
Integer.parseInt(userChoice)!=4) {
                    // checks if user selects from the available menu options
                    if (Integer.parseInt(userChoice) > 4 ||
Integer.parseInt(userChoice) < 1) {
                        System.out.println("Please select an integer between
1-4 only");
                    } else { // if user selects a valid menu option performs
this block of code

                        // stores integer to be added/subtracted
                        int value = 0;
                        if (Integer.parseInt(userChoice) == 1) {
                            System.out.println("Enter value to add: ");
```

```
                            value = Integer.parseInt(typed.readLine());
                        } else if (Integer.parseInt(userChoice) == 2) {
                            System.out.println("Enter value to subtract: ");
                            value = Integer.parseInt(typed.readLine());
                        }
                        System.out.println("Enter your ID: ");
                        // stores ID
                        int id = Integer.parseInt(typed.readLine());
                        // checks if ID range is valid else throws exception
and displays
                        // message to user
                        if (id > 1999 || id < 1000) {
                            throw new IDOutOfRangeException("ID out of range.
Valid range: 1000-1999");
                        }
                        /*
                         * The id, operation and value entered by the user is
sent
                         * to the getResult method. The method returns the
result
                         * of the operation which is stored in result
variable
                         * */
                        int result = getResult(id,
Integer.parseInt(userChoice), value);
                        System.out.println("The result is " + result + ".");
                    }
                }
            } catch (IDOutOfRangeException e) {
                // To catch incorrect range of ID
                System.out.println(e.getMessage());
            } catch (IOException e) {
                // to catch errors when there is an input-output exception
                System.out.println("IO: " + e.getMessage());
            }
        } while (userChoice!=null && !userChoice.equals("4"));
        // exit loop when user enters 4
        System.out.println("Client side quitting.The remote variable server
is still running.");
    }

    /*
     * This method is used to create connection with the server and
     * pass the data to the server to perform the addition/subtraction
     * or return the sum stored against the ID entered by the user. The
client
     * Socket is initialized with source and destination. The BufferedReader
in
     * is used to read data sent from the server and the PrintWriter out is
used to
     * write into the connection stream. The payload is written into the
connection
     * stream using the PrintWriter out. On using out.flush() the data in the
stream
     * is sent to the server to perform the logic. The reply sent by the
server is
     * read using the BufferedReader in and the result is returned to the
```

```
calling method,
     * i.e., the main method
     * */
    public static int getResult(int id, int userChoice, int value) throws
IOException {
        // Code from EchoClientTCP.java in Project 2
        // client socket declared
        Socket clientSocket = null;
        try {
            // server port number
            int serverPort = 7777;
            // client socket initialized
            clientSocket = new Socket("localhost", serverPort);
            // read data from the socket connection
            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            // write data into the socket connection
            PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));
            // stores name of operation based on user choice returned
            // by getOperation method
            String operation= getOperation(userChoice);
            // stores request string to be sent to server
            String payload;
            // get operation does not require value
            if (userChoice==3) {
                payload =  id + " " + operation;
            } else {
                payload =  id + " " + operation + " " + value;
            }
            // write into connection
            out.println(payload);
            // send data written to server
            out.flush();
            String data = in.readLine(); // read a line of data from the
stream
            return Integer.parseInt(data);
        } catch (SocketException e) {
            // to catch errors when errors occur with the network
            System.out.println("Socket: " + e.getMessage());
        } finally {
            // close socket connection
            if(clientSocket!=null) clientSocket.close();
        }
        return -1;
    }

    // returns operation name based on the user choice
    private static String getOperation(int userChoice) {
        if(userChoice==1) {
            return "add";
        } else if(userChoice==2) {
            return "subtract";
        } else {
            return "get";
        }
    }
```

```
    // displays menu options to the user
    public static void displayMenu() {
        String [] menu = {"Add a value to your sum.","Subtract a value from
your sum.",
                "Get your sum","Exit client"};
        for(int i =0; i<menu.length; i++) {
            System.out.printf("%d. %s%n",i+1,menu[i]);
        }
    }

    // New Exception created to track ID which is out of range
    public static class IDOutOfRangeException extends Exception {

        public IDOutOfRangeException(String message) {
            super(message);
        }
    }


}
```

14. Project2Task4Server

```
/*
 * @author: Shivani Poovaiah Ajjikutira
 * Last Modified: 9th October 2021
 *
 * This code follows Task 3 but uses TCP instead of UDP for the data
 * transmission between the client and server.
 * The following code is the server side for a program that returns a
 * number stored against a particular ID or adds/subtracts integers to
 * that integer. The sum/ difference is then stored against the ID.
 * The user id, operation and value(in case of add/subtract) are sent by
 * the client and output of the corresponding operation is returned to the
 * client. The server socket is initialized and continues to listen to any
 * request sent by client sockets connected to port number 7777.The server
 * receives the data from the client through Scanner "in" via the socket
connection
 * formed. The PrintWriter "out" is used to write into the stream and send
data
 * back to the requesting client. The performOperations method checks the
operation
 * passed and does the required logic. A HashMap is used to store the integer
 * corresponding to each ID. In case of addition and subtraction the HashMap
 * values are updated, in case of get, the value for id as key is returned
 * as result to the client. The server is always running.
 * */

import java.io.BufferedWriter;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
```

```java
import java.net.Socket;
import java.net.SocketException;
import java.util.HashMap;
import java.util.Scanner;

public class RemoteVariableServerTCP {
    // maps key-value pair: ID is the key, sum is the value
    static HashMap<Integer,Integer> userIdSums = new HashMap<>();
    public static void main(String[] args){
        // Code from EchoServerTCP.java in Project 2
        System.out.println("Server started");
        // client socket declared
        Socket clientSocket = null;
        try{
            int serverPort = 7777; // the server port number

            // Create a new server socket with port number 7777
            ServerSocket listenSocket = new ServerSocket(serverPort);

            // Since server is always running and listens for requests
            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.
                 */
                if(clientSocket==null || clientSocket.getInputStream().read()
== -1)
                    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())));
                int result;
                // read from client socket
                String data = in.nextLine();
                // split using " "
                String [] requestItems = data.split(" ");
                // result stores sum returned by performOperations method
                result = performOperations(requestItems);
                System.out.printf("Client ID:%s,Operation:%s,Returning sum of
%d\n%n",requestItems[0],requestItems[1],result);
                // write to client socket
                out.println(result);
                // send data written to client socket
                out.flush();
            }
        } catch (SocketException e) {
            // to catch errors when errors occur with the network
            System.out.println("Socket: " + e.getMessage());
```

```
        }catch (IOException e){
            // to catch errors when there is an input-output exception
            System.out.println("IO: " + e.getMessage());
        }
    }

    /*
     * This method performs all the logical operation based on the
     * operation selected by the user as sent by the client in the
     * request. The userIdSums HashMap is used to store the key-value
     * pairs of id and sum. If a particular id does not exist and the
     * user selects "get" operation, put id and value 0 to the hashmap.
     * For any other operation, put id and the value sent from client.
     * If the id is present, then return the value stored against id in
     * the Hashmap in case the operation is "get". In case of
"add"/"subtract"
     * operations add/subtract the value with the existing value in the
     * HashMap, update the value with the new sum/difference and return
     * the new sum/difference
     * */
    public static int performOperations(String [] requestItems) {
        int id = Integer.parseInt(requestItems[0]);
        String operation = requestItems[1];
        // if id not present and operation is "get"
        if(userIdSums.get(id) == null && operation.equals("get")) {
            userIdSums.put(id,0);
        } else if(userIdSums.get(id) == null && !operation.equals("get")) {
            userIdSums.put(id,Integer.parseInt(requestItems[2]));
        }
        // if id present
        else {
            if(operation.equals("add")) {
                int value = Integer.parseInt(requestItems[2]);
                int newTotal = userIdSums.get(id)+value;
                // update new total
                userIdSums.put(id,newTotal);
            } else if(operation.equals("subtract")) {
                int value = Integer.parseInt(requestItems[2]);
                int newTotal = userIdSums.get(id)-value;
                // update new total
                userIdSums.put(id,newTotal);
            }
        }
        // return latest value stores against id
        return userIdSums.get(id);
    }
}
```

15. Project2Task4ClientScreen

```
RemoteVariableServerTCP    RemoteVariableClientTCP
"C:\Program Files\Eclipse Foundation\jdk-16.0.2.7-hot
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:
1001
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:
2
Enter your ID:
1001
The result is 8.
```

```
RemoteVariableServerTCP    RemoteVariableClientTCP
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:
1001
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:
100
Enter your ID:
1002
The result is 100.
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum
4. Exit client
2
```

```
Enter value to subtract:
5
Enter your ID:
1002
The result is 95.
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:
1002
The result is 95.
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum
4. Exit client
1
Enter value to add:
15
Enter your ID:
1003
The result is 15.
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum
4. Exit client
2
Enter value to subtract:
13
Enter your ID:
1003
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:
1003
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.

Process finished with exit code 0
```

RemoteVariableServerTCP RemoteVariableClientTCP

```
"C:\Program Files\Eclipse Foundation\jdk-16.0.2.7-hotspot\bin\java
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:
1001
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:
1002
The result is 95.
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:
1003
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.

Process finished with exit code 0
```

16. Project2Task4ServerScreen

```
RemoteVariableServerTCP    RemoteVariableClientTCP
"C:\Program Files\Eclipse Foundation\jdk-16.0.2.7-hotspot\bin\java.e
Server started
Client ID:1001,Operation:add,Returning sum of 10

Client ID:1001,Operation:subtract,Returning sum of 8

Client ID:1001,Operation:get,Returning sum of 8

Client ID:1002,Operation:add,Returning sum of 100

Client ID:1002,Operation:subtract,Returning sum of 95

Client ID:1002,Operation:get,Returning sum of 95

Client ID:1003,Operation:add,Returning sum of 15

Client ID:1003,Operation:subtract,Returning sum of 2

Client ID:1003,Operation:get,Returning sum of 2
```

```
Client ID:1001,Operation:get,Returning sum of 8

Client ID:1002,Operation:get,Returning sum of 95

Client ID:1003,Operation:get,Returning sum of 2
```

17. Project2Task5Client

```
/*
 * @author: Shivani Poovaiah Ajjikutira
 * Last Modified: 9th October 2021
 *
 * This code follows Task 4 but the client ID is generated using e+n
 * and the data sent to the server is concatenated with the signature
 * of the data to ensure the data received at the server is the right
 * data. The following code is the client side for a program that returns a
 * number stored against a particular ID or adds/subtracts integers to
 * that integer. The sum/ difference is then stored against the ID.
 * The user id, operation and value(in case of add/subtract) are taken
 * from the user and output is the result of the corresponding operation.
 * The client socket is initialized and forms a connection with the server
 * socket having port number 7777.The client passes the data entered by the
 * user to the server via the socket connection formed. The operation logic
 * happens in the server. The program runs till the user chooses option 4,
 * i.e, exit. When the user enters 4 the connection is terminated,however,
 * the server continues running.
 * */

import java.math.BigInteger;
import java.net.*;
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Random;

public class SigningClientTCP {

    public static void main(String [] args) {
        // stores user choice from menu
        String userChoice;
        try {
            // to read user input from console
            BufferedReader typed = new BufferedReader(new
InputStreamReader(System.in));
            // generate public key,private key and store in keys array
            BigInteger [] keys = generateRSAKeys();
            // generate ID using keys and store in id
            String id = generateID(keys);
            /*
             * do-while loop used since we need to run the loop at least once
to show the
             * menu and then continues looping until user choose option 4
             * */
            do {
                // displays menu options
                displayMenu();
                // checks if user input it is not null and not option 4, i.e,
Exit
                if ((userChoice = typed.readLine()) != null &&
Integer.parseInt(userChoice)!=4) {
```

```
                    // checks if user selects from the available menu options
                    if (Integer.parseInt(userChoice) > 4 ||
Integer.parseInt(userChoice) < 1) {
                        System.out.println("Please select an integer between
1-4 only");
                    } else { // if user selects a valid menu option performs
this block of code

                        // stores integer to be added/subtracted
                        int value = 0;
                        if (Integer.parseInt(userChoice) == 1) {
                            System.out.println("Enter value to add: ");
                            value = Integer.parseInt(typed.readLine());
                        } else if (Integer.parseInt(userChoice) == 2) {
                            System.out.println("Enter value to subtract: ");
                            value = Integer.parseInt(typed.readLine());
                        }
                        /*
                         * The id generated, operation,value and keys
generated are sent
                         * to the getResult method. The method returns the
result
                         * of the operation which is stored in result
variable
                         * */
                        int result = getResult(id,
Integer.parseInt(userChoice), value, keys);
                        System.out.println("The result is " + result + ".");
                    }
                }
            } while (userChoice != null && Integer.parseInt(userChoice)!=4);
            // exit loop when user enters 4
            System.out.println("Client side quitting.The remote variable
server is still running.");
        } catch (IOException e) {
            // to catch errors when there is an input-output exception
            System.out.println("IO: " + e.getMessage());
        }
    }

    // this method generates id by hashing e+n
    private static String generateID(BigInteger [] keys) {
        //From project 2 ShortMessageSign.java and RSAExample.java code
        BigInteger e = keys[0]; // n is the modulus for both the private and
public keys
        BigInteger n = keys[2]; // e is the exponent of the public key
        // concatenating e and n
        String pubKey = e.toString()+n.toString();
        try {
            // converting string to byte array
            byte[] bytesOfPublicKey =
pubKey.getBytes(StandardCharsets.UTF_8);
            // Java MessageDigest - to hash strings using SHA-256
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            // stores hashed output e+n
            byte[] hashedOutput = md.digest(bytesOfPublicKey);
```

```
            // we only want last 20 bytes of the hash for id.
            // copy last 20 bytes of hashed bytes to idBytes byte array
            byte[] idBytes = new byte[20];
            for(int i=1; i<idBytes.length;i++) {
                idBytes[i] = hashedOutput[hashedOutput.length-(i)];
            }

            // From the hashed idBytes, create a BigInteger
            BigInteger id = new BigInteger(idBytes);

            // return this as a BigInteger string
            return id.toString();
        } catch (NoSuchAlgorithmException i) {
            // to catch error due to MessageDigest
            System.out.println("ID generation: "+ i.getMessage());
        }
        return null;
    }

    // this method generates public and private keys
    private static BigInteger[] generateRSAKeys() {
        // Code from RSAExample.java provided in Project2

        // 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

        Random rnd = new Random();

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

        // Modulus for both keys
        System.out.printf(" RSA Public key: (e,n) = %d, %d\n",e,n);  // Step
```

```
6: (e,n) is the RSA public key
        System.out.printf(" RSA Private key: (d,n) = %d, %d\n",d,n);  // Step
7: (d,n) is the RSA private key
        return new BigInteger [] {e,d,n};
    }

    /*
     * This method is used to create connection with the server and
     * pass the data to the server to perform the addition/subtraction
     * or return the sum stored against the ID generated. The client
     * Socket is initialized with source and destination. The BufferedReader
in
     * is used to read data sent from the server and the PrintWriter out is
used to
     * write into the socket. The data is concatenated with the signature to
form
     * the string payload. The payload is written into the socket using the
PrintWriter
     * out. On using out.flush() the data in the stream is sent to the server
to perform
     * the logic. The reply sent by the server is read using the
BufferedReader in and the
     * result is returned to the calling method,i.e., the main method
     * */
    public static int getResult(String id, int userChoice, int value,
BigInteger[] keys) throws IOException {
        // Code from EchoClientTCP.java in Project 2
        // client socket declared
        Socket clientSocket = null;
        try {
            // server port number
            int serverPort = 7777;
            // client socket initialized
            clientSocket = new Socket("localhost", serverPort);
            // read data from socket
            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            // write data to socket
            PrintWriter out = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(clientSocket.getOutputStream())));
            // stores name of operation based on user choice returned
            // by getOperation method
            String operation= getOperation(userChoice);
            String tokens;
            // get operation does not require value
            if (userChoice==3) {
                tokens = id+" "+keys[0]+" "+keys[2]+" "+operation;
            } else {
                tokens = id+" "+keys[0]+" "+keys[2]+" "+operation+" "+value;
            }
            // stores encrypted hashed tokens
            String signature = getSignature(tokens,keys);
            // request string to send to server
            String payload = tokens + " " + signature;
            // write into socket
            out.println(payload);
            // send data written to server socket
```

```
            out.flush();
            String data = in.readLine(); // read a line of data from the
stream
            return Integer.parseInt(data);
        } catch (SocketException e) {
            // to catch errors when errors occur with the network
            System.out.println("Socket: " + e.getMessage());
        } finally {
            // close socket connection
            if(clientSocket!=null) clientSocket.close();
        }
        return -1;
    }

    // this method creates a signature of tokens using private key
    private static String getSignature(String tokens, BigInteger[] keys) {
        try {
        // Code from Project 2 ShortMessageSign.java

        // converting string to byte array
        byte[] bytesOfMessage = tokens.getBytes(StandardCharsets.UTF_8);
        // Java MessageDigest - to hash strings using SHA-256
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        // byte array that stores hashed output
        byte[] bigDigest = md.digest(bytesOfMessage);

        // 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];
        // set first index of messageDigest to 0
        messageDigest[0] = 0;
        // copy content of bigDigest to messageDigest byte array
        System.arraycopy(bigDigest, 0, messageDigest, 1, messageDigest.length
- 1);
        BigInteger d = keys[1]; // d is the exponent of the private key
        BigInteger n = keys[2];  // n is the modulus for both the private and
public keys
        // 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();
        } catch (NoSuchAlgorithmException e) {
            // to catch error due to MessageDigest
            System.out.println("Signature creation: "+ e.getMessage());
        }
        return null;
    }

    // returns operation name based on the user choice
    private static String getOperation(int userChoice) {
        if(userChoice==1) {
            return "add";
        } else if(userChoice==2) {
```

```
            return "subtract";
        } else {
            return "get";
        }
    }

    // displays menu options to the user
    public static void displayMenu() {
        String [] menu = {"Add a value to your sum.","Subtract a value from
your sum.",
                "Get your sum","Exit client"};
        for(int i =0; i<menu.length; i++) {
            System.out.printf("%d. %s%n",i+1,menu[i]);
        }
    }
}
```

18. Project2Task5Server

```
19.     /*
    * @author: Shivani Poovaiah Ajjikutira
    * Last Modified: 9th October 2021
    *
    * This code follows Task 4 but the client ID is generated using e+n
    * and the data received from the client is concatenated with the
   signature
    * of the data to ensure the data received is the right data.
    * The following code is the server side for a program that returns a
    * number stored against a particular ID or adds/subtracts integers to
    * that integer. The sum/ difference is then stored against the ID.
    * The operations are performed only if the hashed ID is correct and
    * if the signature is verified. If not an error message is sent back.
    * The user id, operation and value(in case of add/subtract) are sent by
    * the client and output of the corresponding operation is returned to the
    * client. The server socket is initialized and continues to listen to any
    * request sent by client sockets connected to port number 7777.The server
    * receives the data from the client through Scanner "in" via the socket
   connection
    * formed. The PrintWriter "out" is used to write into the stream and send
   data
    * back to the requesting client. The performOperations method checks the
   operation
    * passed and does the required logic. A HashMap is used to store the
   integer
    * corresponding to each ID. In case of addition and subtraction the
   HashMap
    * values are updated, in case of get, the value for id as key is returned
    * as result to the client. The server is always running.
    * */

   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;
```

```java
import java.net.SocketException;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.HashMap;
import java.util.Scanner;

public class VerifyingServerTCP {
    // maps key-value pair: ID is the key, sum is the value
    static HashMap<BigInteger,Integer> userIdSums = new HashMap<>();
    public static void main(String[] args){
        // Code from EchoServerTCP.java in Project 2
        System.out.println("Server started");
        // client socket declared
        Socket clientSocket = null;
        try{
            int serverPort = 7777; // the server port number

            // Create a new server socket with port number 7777
            ServerSocket listenSocket = new ServerSocket(serverPort);

            // Since server is always running and listens for requests
            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.
                 */
                if(clientSocket==null)
                    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())));
                int result;
                // if Scanner in has data from client socket then perform
following block
                if(in.hasNextLine()) {
                    // read data from client socket
                    String data = in.nextLine();
                    // split data using " "
                    String[] requestItems = data.split(" ");
                    // boolean storing ID is correct or not
                    boolean rightIDHash = checkIDHash(requestItems[0],
requestItems[1], requestItems[2]);
                    // boolean storing signature is verified or not
                    boolean rightSignature = checkSignature(requestItems);
                    System.out.printf("Visitor public key(e,n):
(%s,%s)\n", requestItems[1],requestItems[2]);
```

```
                    System.out.printf("Signature verified: %s\n",
rightSignature);
                    System.out.printf("Operation requested: %s\n",
requestItems[3]);
                    // if ID is right and signature is verified perform
operation and send result
                    if (rightIDHash && rightSignature) {
                        // result stores sum returned by performOperations
method
                        result = performOperations(requestItems);
                        System.out.printf("Value returned: %s\n%n%n",
result);
                        // write data to client socket
                        out.println(result);
                    } else {
                        // if either id is incorrect or signature is not
verified return the string below
                        System.out.print("Error in request\n");
                        out.println("Error in request\n");
                    }
                    // send data to client socket
                    out.flush();
                } else {
                    // if Scanner in has no data written into it continue
listening
                    // wait for new request from client
                    clientSocket = listenSocket.accept();
                }
            }
        }catch (SocketException e) {
            // to catch errors when errors occur with the network
            System.out.println("Socket: " + e.getMessage());
        }catch (IOException e){
            // to catch errors when there is an input-output exception
            System.out.println("IO: " + e.getMessage());
        } catch (Exception e) {
            // catch any other exception
            System.out.println("Exception: " + e.getMessage());
        }
    }

    // method to verify the signature
    private static boolean checkSignature(String[] requestItems) throws
Exception {

        // Code from Project 2 ShortMessageVerify.java
        String signature = requestItems[requestItems.length-1]; //
signature received
        BigInteger e = new BigInteger(requestItems[1]); // e is the
exponent of the public key
        BigInteger n = new BigInteger(requestItems[2]); // n is the
modulus for both the private and public keys
        StringBuilder tokens= new StringBuilder();
        // generate the initial data without the signature concatenated
        for(int i=0; i<requestItems.length-1;i++) {
            tokens.append(requestItems[i]).append(" ");
        }
```

```java
        // convert tokens to byte array
        byte[] bytesOfMessageToCheck =
tokens.toString().trim().getBytes(StandardCharsets.UTF_8);
        // clearing StringBuilder
        tokens.setLength(0);

        // compute the digest of the bytesOfMessageToCheck byte array with
SHA-256
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] messageToCheckDigest = md.digest(bytesOfMessageToCheck);

        // messageToCheckDigest is a full SHA-256 digest
        // Add a zero byte since signature is always for positive value
        byte[] newMessage = new byte[messageToCheckDigest.length+1];
        newMessage[0] = 0;
        System.arraycopy(messageToCheckDigest, 0, newMessage, 1,
newMessage.length - 1);

        // converting byte array to BigInteger
        BigInteger bigIntegerToCheck = new BigInteger(newMessage);


        // Take the encrypted hashed string and make it a big integer
        BigInteger encryptedHash = new BigInteger(signature);
        // Decrypt it using public key
        BigInteger decryptedHash = encryptedHash.modPow(e, n);
        // return how the two compare
        return bigIntegerToCheck.compareTo(decryptedHash) == 0;
    }

    // method to check the id
    private static boolean checkIDHash(String id, String e, String n) {
        // concatenating e and n
        String pubKey = e+n;
        try {
            //From project 2 ShortMessageSign.java and RSAExample.java
code

            // Java MessageDigest - to hash strings using SHA-256
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            // converting string to byte array and stores hashed output
e+n in hashedPublicKey
            byte[] hashedPublicKey =
md.digest(pubKey.getBytes(StandardCharsets.UTF_8));

            // we only want last 20 bytes of the hash for id.
            // copy last 20 bytes of hashed bytes to idBytes byte array
            byte[] hashedPubKey = new byte[20];
            for (int i = 1; i < hashedPubKey.length; i++) {
                hashedPubKey[i] = hashedPublicKey[hashedPublicKey.length -
(i)];
            }
            // BigInteger having 20 bytes of hashed (e+n)
            BigInteger computedId = new BigInteger(hashedPubKey);
            // BigInteger having id
            BigInteger bigId = new BigInteger(id);
            // compare computed id and id
```

```
            if(bigId.equals(computedId)) return true;
        } catch (NoSuchAlgorithmException i) {
            // to catch error due to MessageDigest
            System.out.println("ID generation: "+ i.getMessage());
        }
        return false;
    }

    /*
     * This method performs all the logical operation based on the
     * operation selected by the user as sent by the client in the
     * request. The userIdSums HashMap is used to store the key-value
     * pairs of id and sum. If a particular id does not exist and the
     * user selects "get" operation, put id and value 0 to the hashmap.
     * For any other operation, put id and the value sent from client.
     * If the id is present, then return the value stored against id in
     * the Hashmap in case the operation is "get". In case of
"add"/"subtract"
     * operations add/subtract the value with the existing value in the
     * HashMap, update the value with the new sum/difference and return
     * the new sum/difference
     * */
    public static int performOperations(String [] requestItems) {
        BigInteger id = new BigInteger(requestItems[0]);
        String operation = requestItems[3];
        // if id not present and operation is "get"
        if(userIdSums.get(id) == null && operation.equals("get")) {
            userIdSums.put(id,0);
        } else if(userIdSums.get(id) == null && !operation.equals("get"))
{
            userIdSums.put(id,Integer.parseInt(requestItems[4]));
        }
        // if id present
        else {
            if(operation.equals("add")) {
                int value = Integer.parseInt(requestItems[4]);
                int newTotal = userIdSums.get(id)+value;
                // update new total
                userIdSums.put(id,newTotal);
            } else if(operation.equals("subtract")) {
                int value = Integer.parseInt(requestItems[4]);
                int newTotal = userIdSums.get(id)-value;
                // update new total
                userIdSums.put(id,newTotal);
            }
        }
        // return latest value stores against id
        return userIdSums.get(id);
    }
}
```

20. Project2Task5ClientScreen

```
SigningClientTCP    VerifyingServerTCP
"C:\Program Files\Eclipse Foundation\jdk-16.0.2.7-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.1\l
 RSA Public key: (e,n) = 65537, 45142198802407718015418020214292267266965339141995434718085291101504653021188621731046947950314907717 81
 RSA Private key: (d,n) = 17494263777462368160818879066826296657558107382522850465690071614447032629387195244291474801136732322459217 77
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:
5
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
3
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
4
Client side quitting.The remote variable server is still running.

Process finished with exit code 0
```

21. Project2Task5ServerScreen

```
SigningClientTCP    VerifyingServerTCP
"C:\Program Files\Eclipse Foundation\jdk-16.0.2.7-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2021.2.1\lib\idea_rt.jar=55
Server started
Visitor public key(e,n): (65537,48909863019465430953165795087813833578555518540068317499776397469436813910399994940379231966792207647729678956321948100 2
Signature verified: true
Operation requested: add
Value returned: 10


Visitor public key(e,n): (65537,48909863019465430953165795087813833578555518540068317499776397469436813910399994940379231966792207647729678956321948100 2
Signature verified: true
Operation requested: subtract
Value returned: 5


Visitor public key(e,n): (65537,48909863019465430953165795087813833578555518540068317499776397469436813910399994940379231966792207647729678956321948100 2
Signature verified: true
Operation requested: get
Value returned: 5
```