Project2

Created	@October 6, 2022 11:50 AM
Tags	

Task 0

- **Project2Task0Client**

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

public class EchoClientUDP{
    public static void main(String args[]){
        // args give message contents and server hostname.
        // create the socket between two parties on the network.
        DatagramSocket aSocket = null;
        try {
            System.out.println("The client is running.");
            // determine the IP address of a host from the given host's name.
            // InetAddress aHost = InetAddress.getByName(args[0]);
            InetAddress aHost = InetAddress.getByName("localhost");
            // a socket is bound to a port number so that the UDP layer can identify t
he application where data is sent.
            //int serverPort = 6789;
            System.out.println("Enter a port:");
            Scanner sc = new Scanner(System.in);
            int serverPort = sc.nextInt();
            // create a datagram socket and bind to any available port on local machin
e.
            aSocket = new DatagramSocket();
            // use buffer reader to get user's input.
            String nextLine;
            BufferedReader typed = new BufferedReader(new InputStreamReader(System.i
n));
            while ((nextLine = typed.readLine()) != null) {
                // extract byte array from the input string
                byte [] m = nextLine.getBytes();
                // facilitate connectionless transfer of messages from one system to a
nother.
                DatagramPacket request = new DatagramPacket(m,  m.length, aHost, serve
rPort);
                // send a datagram packet(contains data to be sent) from the socket
                aSocket.send(request);
                // receive datagram packet from socket (contains IP addr and port num
 of sender's machine
                byte[] buffer = new byte[1000];
                DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
                aSocket.receive(reply);
```

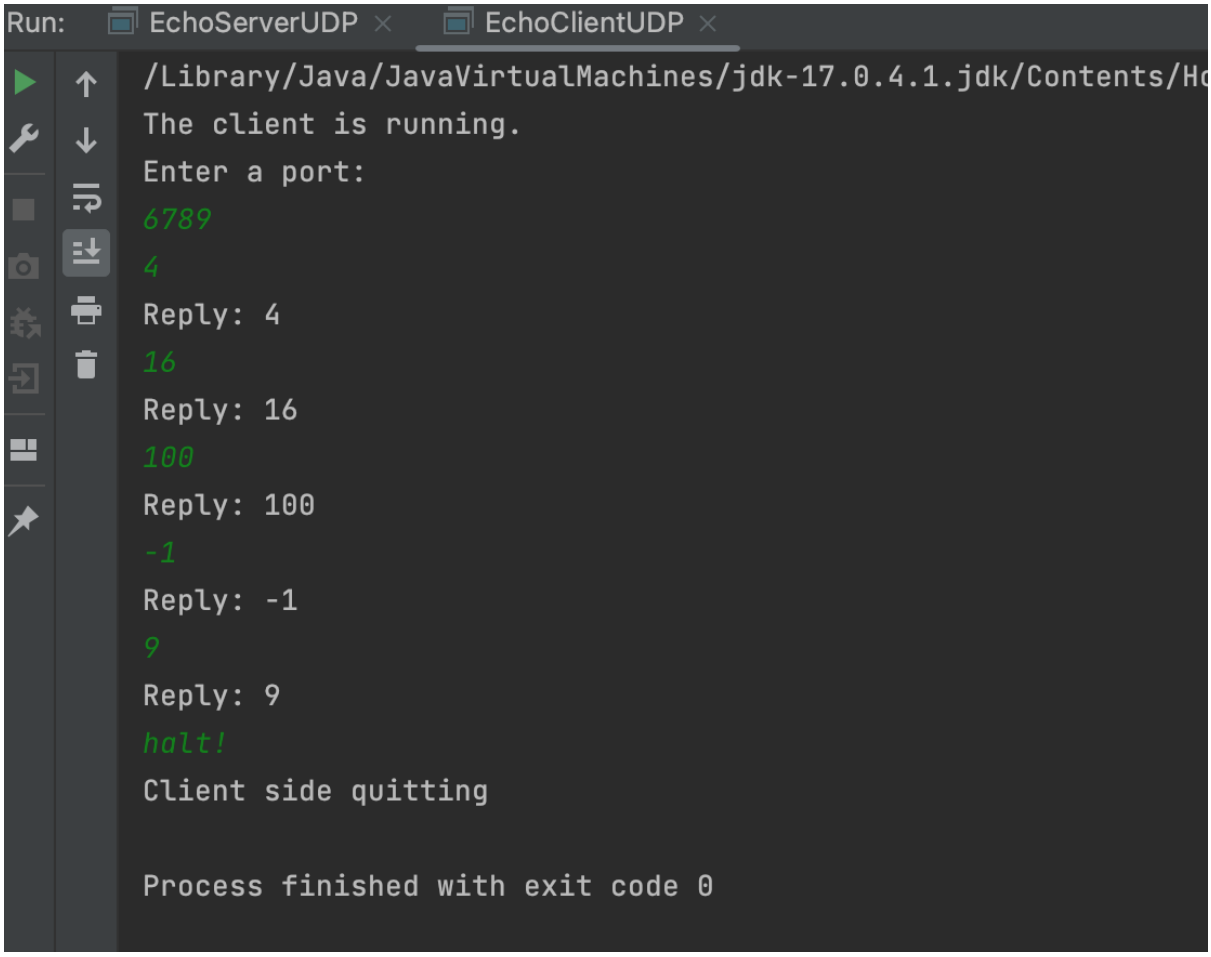
```
                String replyString = new String(reply.getData()).substring(0, reply.ge
tLength()); // reply.getLength()
                if (replyString.equals("halt!")) {
                    System.out.println("Client side quitting");
                    break;
                }
                System.out.println("Reply: " + replyString);
            }
        // exception handling
        }catch (SocketException e) {System.out.println("Socket: " + e.getMessage());
        }catch (IOException e){System.out.println("IO: " + e.getMessage());
        }finally {if(aSocket != null) aSocket.close();}
    }
}
```

- **Project2Task0Server**

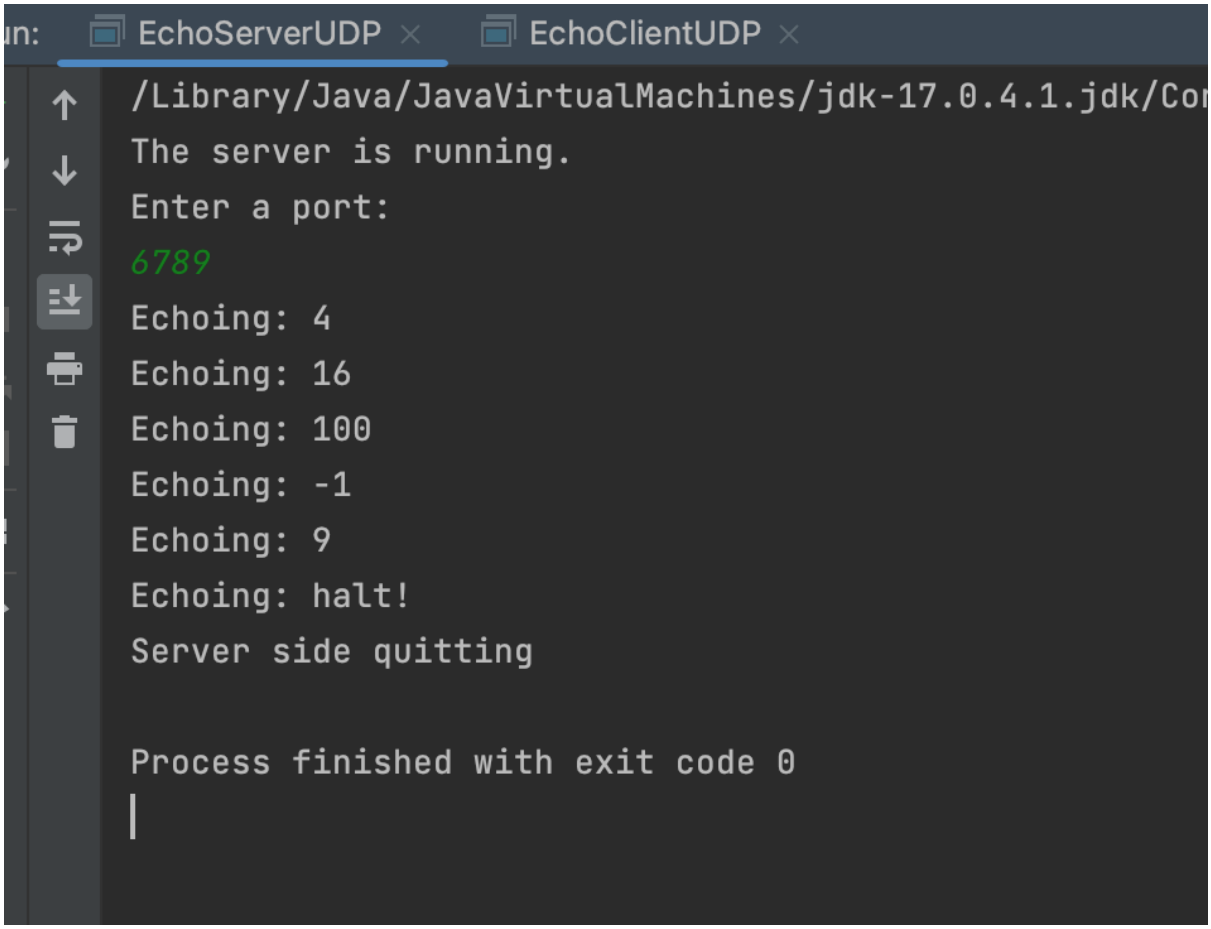
```
import java.net.*;
import java.io.*;
import java.util.Scanner;

public class EchoServerUDP{
    public static void main(String args[]){
        DatagramSocket aSocket = null;
        byte[] buffer = new byte[1000];
        try{
            System.out.println("The server is running.");
            // create a data socket connection to transfer data
            System.out.println("Enter a port:");
            Scanner sc = new Scanner(System.in);
            int serverPort = sc.nextInt(); // 6789
            aSocket = new DatagramSocket(serverPort);
            DatagramPacket request = new DatagramPacket(buffer, buffer.length);
            while(true){
                // for server, first receive what client sent to you.
                aSocket.receive(request);
                // create a datagram packet based on what client sent.
                DatagramPacket reply = new DatagramPacket(request.getData(),
                        request.getLength(), request.getAddress(), request.getPort());
                // output what client sent
                // examine the length of requestString
                // copy request data to an array with correct number of bytes; use thi
s array to build requestString of correct size
                String requestString = new String(request.getData()).substring(0, requ
est.getLength());
                // server: display request arriving from the client.
                System.out.println("Echoing: "+requestString);
                // send back reply packet to client
                aSocket.send(reply);
                if (requestString.equals("halt!")){
                    System.out.println("Server side quitting");
                    break;
                }
            }
            // exception handling
```

```
        }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
        }catch (IOException e) {System.out.println("IO: " + e.getMessage());
        }finally {if(aSocket != null) aSocket.close();}
    }
}
```

- **Project2Task0ClientConsole**



- **Project2Task0ServerConsole**



Task 1

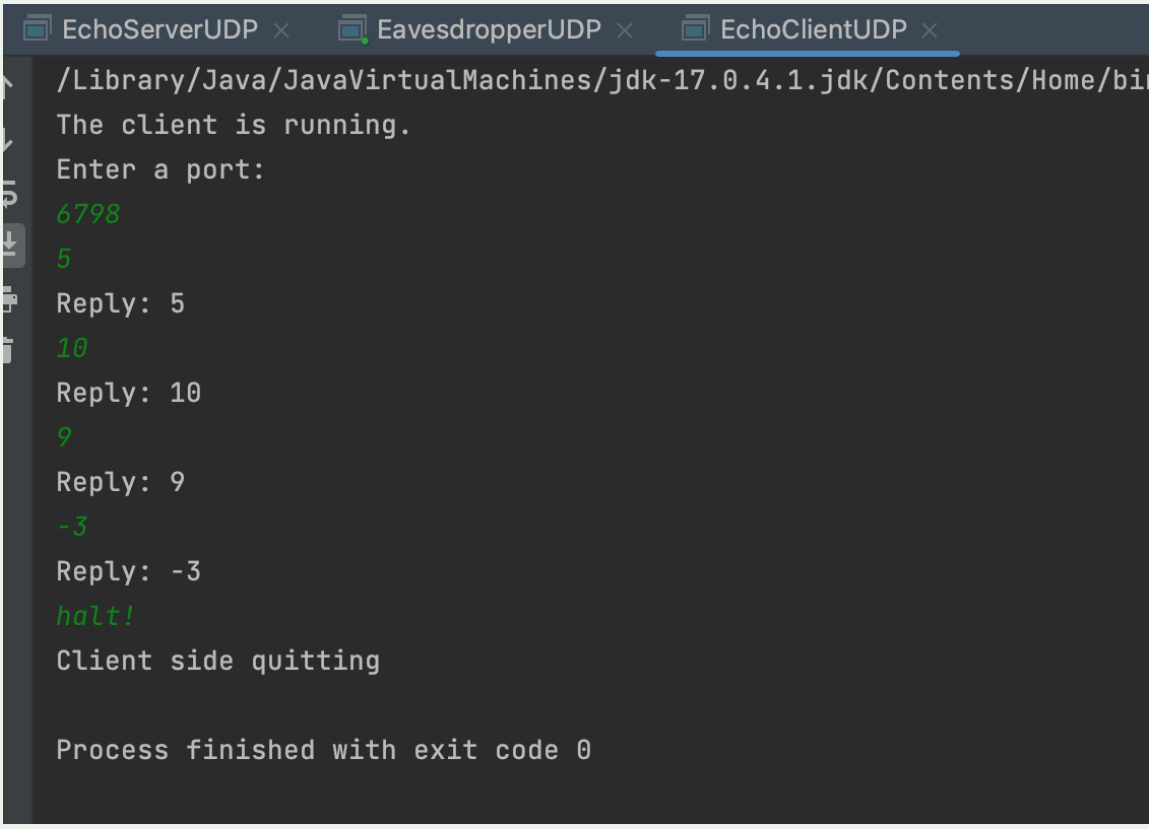
- EavesdropperUDP.java

```
/**
 * @author: Olivia Wu (jingyiw2)
 */

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

public class EavesdropperUDP {
    /**
     * Responsibility: passive malicious player
     * 1. state it is running and ask for two ports: one to listen on and the other th
e server to masquerade as
     * 2. display all msgs going through it.
     * 3. eavesdrop on the wire, masquerading as the server on port 6789
     */
    public static void main(String[] args) {
        DatagramSocket aSocket = null;
        DatagramSocket eavSocket = null;
        byte[] buffer = new byte[1000];
```

```
        try{
            System.out.println("EavesdropperUDP is running.");
            InetAddress aHost = InetAddress.getByName("localhost");

            Scanner sc = new Scanner(System.in);
            System.out.println("Enter a port for EavesdropperUDP to listen on:"); // 6
798
            int eavPort = sc.nextInt();
            System.out.println("Enter a port of server to masquerade as:"); // 6789
            int listenPort = sc.nextInt();

            aSocket = new DatagramSocket();
            eavSocket = new DatagramSocket(eavPort); // socket for fake port
            /*
            receive and deliver to the actual port.
             */
            DatagramPacket request = new DatagramPacket(buffer, buffer.length);
            DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
            while (true){
                // receive msg from client and display
                eavSocket.receive(request);
                String requestString = new String(request.getData()).substring(0, requ
est.getLength());
                // when seeing halt request, write a line of asterisks to its console
 to make an alert
                // eavs does not halt but display to its console and pass the halt on
 to the server
                if (requestString.equals("halt!")){
                    System.out.println("*********************************************
*****************");
                }
                System.out.println("Request from client:"+requestString);


                // send the message to actual server
                DatagramPacket message = new DatagramPacket(request.getData(), reques
t.getLength(), aHost, listenPort);
                aSocket.send(message);

                // receive message from the actual server
                aSocket.receive(reply);
                String receiveString = new String(reply.getData()).substring(0, reply.
getLength());
                System.out.println("Receive from server:"+receiveString);

                // send back to the client
                DatagramPacket message2 = new DatagramPacket(request.getData(), reques
t.getLength(), aHost, request.getPort());
                eavSocket.send(message2);
            }
        } catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        } catch (UnknownHostException e) {
            System.out.println("Host: "+e.getMessage());
        } catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        } finally {
            if (aSocket != null) aSocket.close();
```

```
            if (eavSocket != null) eavSocket.close();
        }
    }
}
```

- **Project2Task1ThreeConsoles**
  - **client:**



```
EchoServerUDP   EavesdropperUDP   EchoClientUDP
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents
The client is running.
Enter a port:
6789
5
Reply: 5
10
Reply: 10
9
Reply: 9
-3
Reply: -3
halt!
Client side quitting

Process finished with exit code 0
```

- **server:**

```
EchoServerUDP   EavesdropperUDP   EchoClientUDP
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents/Home/bi
The server is running.
Enter a port:
6789
Echoing: 5
Echoing: 10
Echoing: 9
Echoing: -3
Echoing: halt!
Server side quitting

Process finished with exit code 0
```

- **eavesdropper:**

```
EchoServerUDP   EavesdropperUDP   EchoClientUDP
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents/Home/bin/jav
EavesdropperUDP is running.
Enter a port for EavesdropperUDP to listen on:
6798
Enter a port of server to masquerade as:
6789
Request from client:5
Receive from server:5
Request from client:10
Receive from server:10
Request from client:9
Receive from server:9
Request from client:-3
Receive from server:-3
*************************************************************
Request from client:halt!
Receive from server:halt!
```

```
EchoServerUDP   EavesdropperUDP   EchoClientUDP
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents/Home/b
EavesdropperUDP is running.
Enter a port for EavesdropperUDP to listen on:
6798
Enter a port of server to masquerade as:
6789
```

Task 2

- **Project2Task2Client**

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

```
/**
 * Client Responsibility:
 * 1. all communication code placed in method "add".
 * 2. no communications code in main.
 * 3. client add: not performing any addition, ask the server do the addition.
 * 4. add is a proxy for the server
 * 5. when calling local add method, RPC
 */
public class AddingClientUDP{
    public static int port = 6789;
    public static InetAddress aHost;
    public static DatagramSocket aSocket;
    public AddingClientUDP(int portNum){
        try {
            aSocket = new DatagramSocket();
            port = portNum;
            aHost = InetAddress.getByName("localhost");
        } catch (SocketException | UnknownHostException e){
            System.out.println("Error:"+e.getMessage());
        }
    }
    /**
     * all socket communication code, not performing any addition
     * request the server to perform addition
     * @param i
     * @return the cumulative sum
     */
    public static int add(int i){
        // get byte array of input int
        byte[] bytes = String.valueOf(i).getBytes();
        try {
            // data packet to send to server
            DatagramPacket request = new DatagramPacket(bytes,  bytes.length, aHost, p
ort);
            aSocket.send(request);
            // data packet to receive
            byte[] buffer = new byte[1000];
            DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
            aSocket.receive(reply);
            String replyResult = new String(reply.getData()).substring(0, reply.getLen
gth());
            return Integer.valueOf(replyResult);
        } catch (IOException e) {
            System.out.println("IO: "+e.getMessage());
        }
        return -1;
    }
    /**
     * call on local add method to make an RPC
     * @param args
     */
    public static void main(String args[]){
        try {
            System.out.println("The client is running.");
            // ask the user to input server port
            System.out.println("Please enter server port: ");
            Scanner sc = new Scanner(System.in);
            port = sc.nextInt();
```

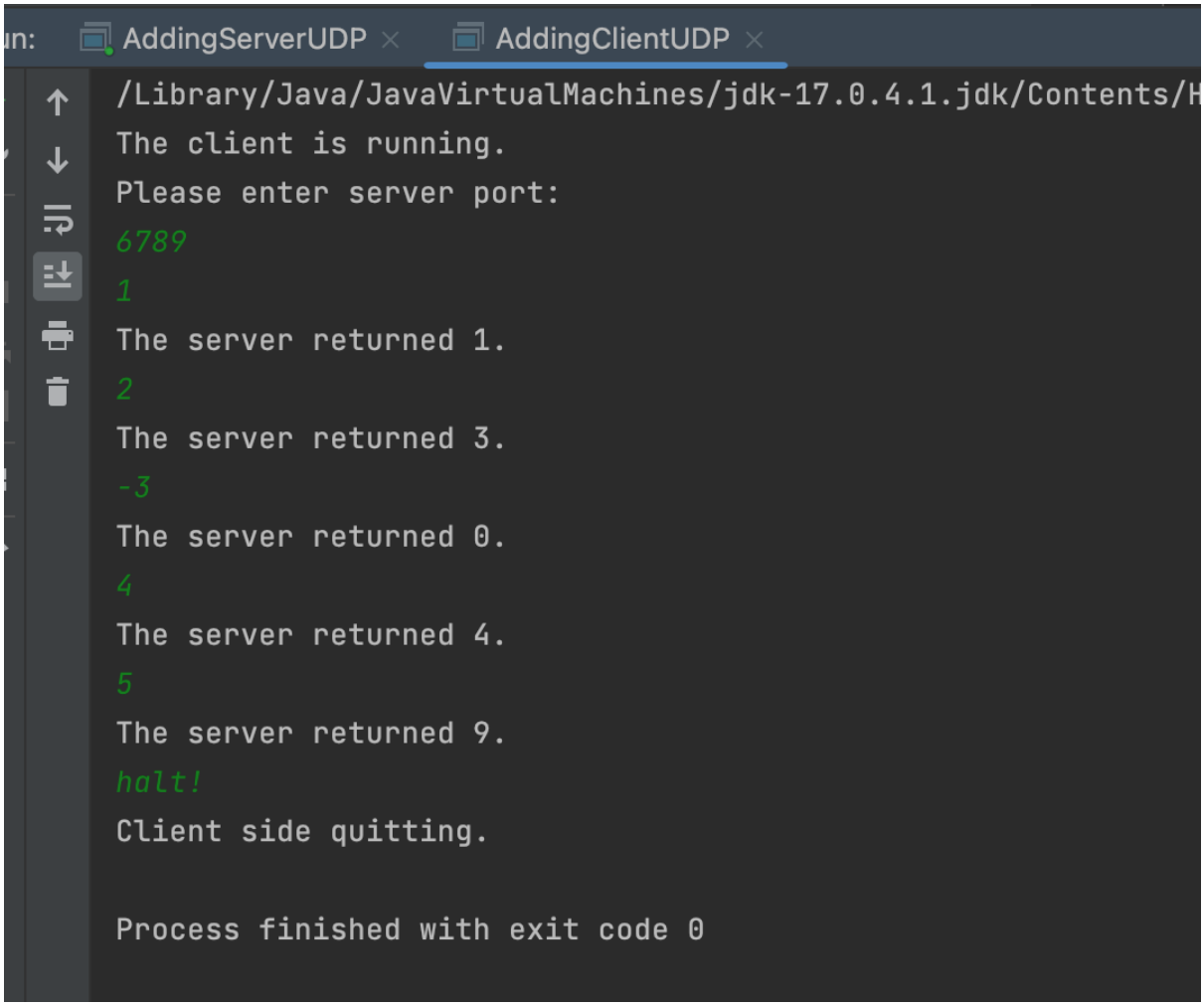
```
            AddingClientUDP clientUDP = new AddingClientUDP(port);
            while (sc.hasNext()) {
                if (sc.nextLine().equals("halt!")){
                    System.out.println("Client side quitting.");
                    break;
                }
                if (sc.hasNextInt()){
                    int addedNum = sc.nextInt();
                    // call add
                    int result = add(addedNum);
                    System.out.println("The server returned "+result+".");
                }
            }
        } finally {if(aSocket != null) aSocket.close();}
    }
}
```

- **Project2Task2Server**

```
import java.net.*;
import java.io.*;
import java.util.Scanner;
/**
 * Server's responsibility:
 * 1. hold an int value sum=0, receive requests from client: an int added to sum.
 * 2. upon each request, response new sum to client.
 * 3. upon each visit, display client's request and new sum.
 * 4. listen for a socket connection code separated from addition operation
 */
public class AddingServerUDP{
    public static int sum = 0;
    public static int port = 6789;
    private static int addNum(int addedNum){
        sum += addedNum;
        return sum;
    }
    public static void main(String args[]){
        System.out.println("Server started.");
        DatagramSocket aSocket = null;
        byte[] buffer = new byte[1000];
        try{
            aSocket = new DatagramSocket(port);
            DatagramPacket request = new DatagramPacket(buffer, buffer.length);
            while(true){
                // get request from client
                aSocket.receive(request);
                String addedNum = new String(request.getData()).substring(0, request.g
etLength());
                System.out.println("Adding "+addedNum+" to "+sum);
                // add number to sum
                sum = addNum(Integer.valueOf(addedNum));
                System.out.println("Returning sum of "+sum+" to client.");
                // reply the client with updated sum
                byte[] bytes = String.valueOf(sum).getBytes();
                DatagramPacket reply = new DatagramPacket(bytes, bytes.length, reques
```

```
t.getAddress(), request.getPort());
                aSocket.send(reply);
            }
        }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
        }catch (IOException e) {System.out.println("IO: " + e.getMessage());
        }finally {if(aSocket != null) aSocket.close();}
    }
}
```

- **Project2Task2ClientConsole**



```
un:  AddingServerUDP ×  AddingClientUDP ×
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Conte
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.

Process finished with exit code 0
```

- **Project2Task2ServerConsole**

```
n:  AddingServerUDP ×   AddingClientUDP ×
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jc
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.
```

Task 3

- **Project2Task3Client**

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

```
import java.util.Scanner;
/**
 * Client Responsibility:
 * 1. request add, subtract, or get(idempotent).
 * 2. each request pass along an int ID to uniquely identify the user.
 * 3. client form a packet: ID, operation (add or subtract or get), and value (if the
 operation is other than get); server do the computation(with ID).
 * 4. menu-driven.
 * 5. repeatedly ask the user for userID, operation, value.
 *  - get: value held on the server is simply returned
 *  - add/subtract: do the operation and return sum
 * 6. display each returned value from server to user.
 * 7. for new ID, sum=0.
 * 8. ID range: 0~999.
 * 9. exit option on menu has no impact on server.
 */
public class RemoteVariableClientUDP{
    private int port = 6789;
    private InetAddress aHost;
    private DatagramSocket aSocket;
    public RemoteVariableClientUDP(int portNum) throws UnknownHostException, SocketExc
eption {
        port = portNum;
        aHost = InetAddress.getByName("localhost");
        aSocket = new DatagramSocket();
    }

    /**
     * output the menu and initialize variables
     */
    public void init(){
        System.out.println("1. Add a value to your sum.\n" +
                    "2. Subtract a value from your sum.\n" +
                    "3. Get your sum.\n" +
                    "4. Exit client");
    }

    /**
     * Generate information request based on user's choice
     * @param operationNum
     * @return data packet of client to be sent
     */
    public String start(int operationNum,  Scanner sc) {
        int option = operationNum;
        int value = 0;
        String valueString = "";
        int id;
        switch (option){
            case 1:
                System.out.println("Enter value to add: ");
                value = sc.nextInt();
                valueString = String.valueOf(value);
                break;
            case 2:
                System.out.println("Enter value to subtract:");
                value = sc.nextInt();
                valueString = String.valueOf(value);
                break;
```

```
            case 4:
                System.out.println("Client side quitting. The remote variable server i
s still running.");
                return "halt";
        }
        System.out.println("Enter your ID: ");
        id = sc.nextInt();
        return String.valueOf(id)+" "+String.valueOf(option)+" "+valueString;
    }

    /**
     * all socket communication code, not performing any operation
     * request the server to perform operation: add, subtract, get
     * @param dataPacket
     * @return the cumulative sum
     */
    public boolean communicate(String dataPacket){
        // get byte array of input int
        byte[] bytes = dataPacket.getBytes();
        try {
            // data packet to send to server
            DatagramPacket request = new DatagramPacket(bytes,  bytes.length, aHost, p
ort);
            aSocket.send(request);
            // data packet to receive
            byte[] buffer = new byte[1000];
            DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
            aSocket.receive(reply);
            String replyResult = new String(reply.getData()).substring(0, reply.getLen
gth());

            if (replyResult.equals("halt")) {
                return false;
            }
            System.out.println("The result is "+replyResult+".");
        } catch (IOException e) {
            System.out.println("IO: "+e.getMessage());
        }
        return true;
    }


    /**
     * call on local add method to make an RPC
     * @param args
     */
    public static void main(String args[]){
        try {
            System.out.println("The client is running.");
            Scanner sc = new Scanner(System.in);
            // ask the user to input server port
            System.out.println("Please enter server port: ");
            int port = sc.nextInt();
            RemoteVariableClientUDP clientUDP = new RemoteVariableClientUDP(port);

            boolean sendFlag = true;
            while (sendFlag) {
                clientUDP.init();
```

```
                int operationNum = sc.nextInt();
                // generate message sent to server
                String operationString = clientUDP.start(operationNum, sc);
                if (operationString.equals("halt")){
                    sendFlag = false;
                    break;
                }
                // send message to server
                sendFlag = clientUDP.communicate(operationString);
            }
            if(clientUDP.aSocket != null) clientUDP.aSocket.close();

        } catch (SocketException e) {
            throw new RuntimeException(e);
        } catch (UnknownHostException e) {
            throw new RuntimeException(e);
        }
    }
}
```

- **Project2Task3Server**

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

/**
 * Server's responsibility:
 * 1. map each ID to value of sum using TreeMap.
 * 2. assume that ID are positive integers.
 *
 */
public class RemoteVariableServerUDP{
    private int sum, id;
    private int port;
    private DatagramSocket aSocket;
    private TreeMap<Integer, Integer> hashMap;
    private DatagramPacket request;
    public RemoteVariableServerUDP() throws SocketException{
        sum = 0;
        port = 6789;
        aSocket = new DatagramSocket(port);
        byte[] buffer = new byte[1000];
        request = new DatagramPacket(buffer, buffer.length);
        hashMap = new TreeMap<>();

    }

    /**
     * add number for each client ID
     * @param id
     * @param value
     * @return the result being returned and printed out
     */
```

```
    public int addNum(int id, int value){
        if (hashMap.containsKey(id)){
            sum = hashMap.get(id)+value;
            hashMap.put(id, sum);
        } else {
            sum = value;
            hashMap.put(id, sum);
        }
        return sum;
    }

    /**
     * subtract number for each clent ID
     * @param id
     * @param value
     * @return the result being returned and printed out
     */
    public int subNum(int id, int value){
        if (hashMap.containsKey(id)){
            sum = hashMap.get(id)-value;
            hashMap.put(id, sum);
        } else {
            sum = 0-value;
            hashMap.put(id, sum);
        }
        return sum;
    }
    public static void main(String args[]){
        System.out.println("Server started.");
        try{
            RemoteVariableServerUDP serverUDP = new RemoteVariableServerUDP();
            while(true){
                // get request from client
                serverUDP.aSocket.receive(serverUDP.request);
                String operationString = new String(serverUDP.request.getData()).subst
ring(0, serverUDP.request.getLength());
                // extract elements from string
                int id = Integer.parseInt(operationString.split(" ")[0]);
                int option = Integer.parseInt(operationString.split(" ")[1]);
                int result = 0;
                int value = 0;
                switch (option){
                    case 1:
                        value = Integer.parseInt(operationString.split(" ")[2]);
                        result = serverUDP.addNum(id, value);
                        break;
                    case 2:
                        value = Integer.parseInt(operationString.split(" ")[2]);
                        result = serverUDP.subNum(id, value);
                        break;
                    case 3:
                        result = serverUDP.hashMap.get(id);
                        break;
                }
                System.out.println("Client ID: "+id+"; Operation Request: "+option+";
 Returned Value: "+result);

                // reply to the client with updated result
```

```
                byte[] bytes = String.valueOf(result).getBytes();
                DatagramPacket reply = new DatagramPacket(bytes, bytes.length, serverU
DP.request.getAddress(), serverUDP.request.getPort());
                serverUDP.aSocket.send(reply);
            }
        }catch (SocketException e){System.out.println("Socket: " + e.getMessage());
        }catch (IOException e) {System.out.println("IO: " + e.getMessage());
        }
    }
}
```

- **Project2Task3ClientConsole**

RemoteVariableServerUDP RemoteVariableClientUDP

```
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents/Home/bin
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:
3
Enter your ID:
102
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
1
Enter value to add:
5
Enter your ID:
110
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:
7
Enter your ID:
120
The result is -7.
```

```
RemoteVariableServerUDP    RemoteVariableClientUDP
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
2
Enter value to subtract:
-10
Enter your ID:
110
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:
4
Enter your ID:
102
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
1
Enter value to add:
6
Enter your ID:
120
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
4
Client side quitting. The remote variable server is still running.

Process finished with exit code 0
```

```
RemoteVariableServerUDP    RemoteVariableClientUDP
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents/Home/bin/java -jav
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:
102
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
3
Enter your ID:
110
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
3
Enter your ID:
120
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
4
Client side quitting. The remote variable server is still running.

Process finished with exit code 0
```

- **Project2Task3ServerConsole**

```
RemoteVariableServerUDP    RemoteVariableClientUDP
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents/Home/bin/java -ja
Server started.
Client ID: 102; Operation Request: 1; Returned Value: 3
Client ID: 110; Operation Request: 1; Returned Value: 5
Client ID: 120; Operation Request: 2; Returned Value: -7
Client ID: 110; Operation Request: 2; Returned Value: 15
Client ID: 102; Operation Request: 2; Returned Value: -1
Client ID: 120; Operation Request: 1; Returned Value: -1
Client ID: 102; Operation Request: 3; Returned Value: -1
Client ID: 110; Operation Request: 3; Returned Value: 15
Client ID: 120; Operation Request: 3; Returned Value: -1
```

Task 4

- **Project2Task4Client**

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

public class RemoteVariableClientTCP {
    private int port = 7777;
    private InetAddress aHost;
    private Socket clientSocket;
    public RemoteVariableClientTCP(int portNum) throws IOException {
        port = portNum;
        aHost = InetAddress.getByName("localhost");
        clientSocket = new Socket(aHost, port);
    }

    /**
     * output the menu and initialize variables
     */
    public void init(){
        System.out.println("1. Add a value to your sum.\n" +
                "2. Subtract a value from your sum.\n" +
                "3. Get your sum.\n" +
                "4. Exit client");
    }

    /**
     * Generate information request based on user's choice
     * @param operationNum
     * @return data packet of client to be sent
     */
```

```
    public String start(int operationNum,  Scanner sc) {
        int option = operationNum;
        int value = 0;
        String valueString = "";
        int id;
        switch (option){
            case 1:
                System.out.println("Enter value to add: ");
                value = sc.nextInt();
                valueString = String.valueOf(value);
                break;
            case 2:
                System.out.println("Enter value to subtract:");
                value = sc.nextInt();
                valueString = String.valueOf(value);
                break;
            case 4:
                System.out.println("Client side quitting. The remote variable server i
s still running.");
                return "halt";
        }
        System.out.println("Enter your ID: ");
        id = sc.nextInt();
        return id +" "+ option +" "+valueString;
    }

    /**
     * all socket communication code, not performing any operation
     * request the server to perform operation: add, subtract, get
     * @param dataPacket
     */
    public void communicate(String dataPacket ) {
        try {
            // send to the server through TCP
            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.
getInputStream()));
            PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWrite
r(clientSocket.getOutputStream())));
            out.println(dataPacket);
            out.flush();
            // receive from server
            String replyResult= in.readLine(); // read a line of data from the stream
            System.out.println("The result is " + replyResult + ".");
        } catch (IOException e) {
            System.out.println("IO Exception:" + e.getMessage());
        }
    }

    /**
     * proxy design
     * @param args
     */
    public static void main(String args[]){
        try {
            System.out.println("The client is running.");
            Scanner sc = new Scanner(System.in);
            // ask the user to input server port
            System.out.println("Please enter server port: ");
```

```
            int port = sc.nextInt(); // 7777
            RemoteVariableClientTCP clientTCP = new RemoteVariableClientTCP(port);

            boolean sendFlag = true;
            while (sendFlag) {
                clientTCP.init();
                int operationNum = sc.nextInt();
                // generate message sent to server
                String operationString = clientTCP.start(operationNum, sc);
                if (operationString.equals("halt")){
                    sendFlag = false;
                    break;
                }
                // send message to server
                clientTCP.communicate(operationString);
            }
        }  catch (IOException e) {
            System.out.println("IO:"+e.getMessage());
        }
    }

}
```

- **Project2Task4Server**

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

public class RemoteVariableServerTCP {
    private int sum, id;
    private TreeMap<Integer, Integer> hashMap;
    private int port; // 7777
    private ServerSocket aSocket;
    private Socket clientSocket;

    public RemoteVariableServerTCP() throws IOException {
        sum = 0;
        port = 7777;
        hashMap = new TreeMap<>();
        aSocket = new ServerSocket(port);
    }
    /**
     * add number for each client ID
     * @param id
     * @param value
     * @return the result being returned and printed out
     */
    public int addNum(int id, int value){
        if (hashMap.containsKey(id)){
            sum = hashMap.get(id)+value;
            hashMap.put(id, sum);
        } else {
            sum = value;
```

```
            hashMap.put(id, sum);
        }
        return sum;
    }

    /**
     * subtract number for each client ID
     * @param id
     * @param value
     * @return the result being returned and printed out
     */
    public int subNum(int id, int value){
        if (hashMap.containsKey(id)){
            sum = hashMap.get(id)-value;
            hashMap.put(id, sum);
        } else {
            sum = 0-value;
            hashMap.put(id, sum);
        }
        return sum;
    }
    public static void main(String args[]){
        System.out.println("Server started.");
        try{
            RemoteVariableServerTCP serverTCP = new RemoteVariableServerTCP();
            serverTCP.clientSocket = serverTCP.aSocket.accept();
            Scanner in = new Scanner(serverTCP.clientSocket.getInputStream());
            PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWrite
r(serverTCP.clientSocket.getOutputStream())));
            String operationString = "";
            while(true){
                // receive from client
                if (serverTCP.clientSocket==null){
                    serverTCP.clientSocket = serverTCP.aSocket.accept();
                    in = new Scanner(serverTCP.clientSocket.getInputStream());
                    out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(se
rverTCP.clientSocket.getOutputStream())));
                }
                try {
                    operationString = in.nextLine();
                } catch(Exception e){
                    serverTCP.clientSocket.close();
                    serverTCP.clientSocket = null;
                    continue;
                }
                // extract elements from string
                int id = Integer.parseInt(operationString.split(" ")[0]);
                int option = Integer.parseInt(operationString.split(" ")[1]);
                int result = 0;
                int value = 0;
                switch (option){
                    case 1:
                        value = Integer.parseInt(operationString.split(" ")[2]);
                        result = serverTCP.addNum(id, value);
                        break;
                    case 2:
                        value = Integer.parseInt(operationString.split(" ")[2]);
                        result = serverTCP.subNum(id, value);
```

```
                        break;
                    case 3:
                        result = serverTCP.hashMap.get(id);
                        break;
                }
                System.out.println("Client ID: "+id+"; Operation Request: "+option+";
 Returned Value: "+result);
                out.println(result+"\n");
                out.flush();
            }
        }catch (IOException e) {System.out.println("IO: " + e.getMessage());
        }
    }
}
```

- **Project2Task4ClientConsole**

RemoteVariableServerTCP RemoteVariableClientTCP

```
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Co
The client is running.
Please enter server port:
7777
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:
102
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:
4
Enter your ID:
102
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:
-10
Enter your ID:
110
The result is 10.
```

```
n:  RemoteVariableServerTCP  RemoteVariableClientTCP
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:
110
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:
7
Enter your ID:
120
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
1
Enter value to add:
6
Enter your ID:
120
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
4
Client side quitting. The remote variable server is still running.

Process finished with exit code 0
```

```
RemoteVariableClientTCP
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents/Home/bin/java -ja
The client is running.
Please enter server port:
7777
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
102
The result is -1.
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:
110
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
3
Enter your ID:
120
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
4
Client side quitting. The remote variable server is still running.

Process finished with exit code 0
```

- **Project2Task4ServerConsole**

```
RemoteVariableServerTCP
/Library/Java/JavaVirtualMachines/jdk-17.0.4.1.jdk/Contents/Home/bin/
Server started.
Client ID: 102; Operation Request: 1; Returned Value: 3
Client ID: 102; Operation Request: 2; Returned Value: -1
Client ID: 110; Operation Request: 2; Returned Value: 10
Client ID: 110; Operation Request: 1; Returned Value: 15
Client ID: 120; Operation Request: 2; Returned Value: -7
Client ID: 120; Operation Request: 1; Returned Value: -1
Client ID: 102; Operation Request: 3; Returned Value: -1
Client ID: 110; Operation Request: 3; Returned Value: 15
Client ID: 120; Operation Request: 3; Returned Value: -1
```

Task 5

- **Project2Task5Client**

```
import java.net.*;
import java.io.*;
import java.util.Scanner;
import java.util.Random;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
/**
 * Client:
 * 1. create new RSA public and private keys and display these keys to the user.
 * 2. client ID: least significant 20 bytes of the hash of public key <e, n> and compu
ted in the client code.
 * 3. transmit public key with each request.
 * 4. clients sign each request. with d and n, encrypt hash of the msg and add to each
request.
 *   - here we use SHA-256 for h():
 *      - last20BytesOf(h(e+n))
 *      - E(h(all prior tokens),d)
 *      - signature is an encrypted hash using d and n < private key
 * 5. RSA-related refer to the link below.
 * @Reference: https://github.com/CMU-Heinz-95702/Project-2-Client-Server
 *
 */
public class SigningClientTCP {
    private int port = 7777;
    private InetAddress host;
```

```
    private Socket clientSocket;
    /*
    public key: (n, e); private key: (n. d)
    n: modulus for both private and public keys
    e: exponent of public key
    d: exponent of private key
     */
    private BigInteger n, e, d;
    private String publicKey, privateKey;
    private String id;
    Random rnd;
    public SigningClientTCP(int portNum) throws IOException {
        // RSA algorithm:
        rnd = new Random();
        // Step 1: Generate two large random primes.
        // We use 400 bits here, but best practice for security is 2048 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.O
NE));
        // 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);
        publicKey = e+","+n;
        privateKey = d+","+n;
        host = InetAddress.getByName("localhost");
        clientSocket = new Socket(host, port);
    }

    /**
     * create id: last20BytesOf(h(e+n))
     * @param en
     */
    public String setId(String en) throws UnsupportedEncodingException, NoSuchAlgorith
mException {
        // compute the digest with SHA-256
        byte[] bytesOfMessage = en.getBytes("UTF-8");
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] bigDigest = md.digest(bytesOfMessage);

        // we only want 20 bytes of the hash for ShortMessageSign
        // we add a 0 byte as the most significant byte to keep value to be signed non
-negative.
        byte[] messageDigest = new byte[21];
        messageDigest[0] = 0;   // most significant set to 0
        // iterate to take bytes from SHA-256
        for (int i=0; i<20; i++){
            messageDigest[20-i] = bigDigest[bigDigest.length-1-i];
        }
        // From the digest, create a BigInteger
        BigInteger m = new BigInteger(messageDigest);
        id = m.toString();
        return m.toString();
```

```
    }

    /**
     * get operation option and number to operate on from user's input and return stri
ng to send
     * @param sc
     * @return string to send
     */
    public String start(Scanner sc){
        System.out.println("1. Add a value to your sum.\n" +
                "2. Subtract a value from your sum.\n" +
                "3. Get your sum.\n" +
                "4. Exit client");
        int option = sc.nextInt();
        int value=0;
        String valueString="";
        switch (option){
            case 1:
                System.out.println("Enter value to add: ");
                value = sc.nextInt();
                valueString = String.valueOf(value);
                break;
            case 2:
                System.out.println("Enter value to subtract:");
                value = sc.nextInt();
                valueString = String.valueOf(value);
                break;
            case 4:
                System.out.println("Client side quitting. The remote variable server i
s still running.");
                return "halt";
        }
        return id +","+ option +","+valueString;
    }

    /**
     * generate signature for requested message: encrypted hash using d and n(private
 key)
     * @param message
     * @return signature string
     * @throws UnsupportedEncodingException
     * @throws NoSuchAlgorithmException
     */
    public String getSignature(String message) throws UnsupportedEncodingException, No
SuchAlgorithmException {
        // compute the digest with SHA-256
        byte[] bytesOfMessage = message.getBytes("UTF-8");
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] bigDigest = md.digest(bytesOfMessage);

        // we only want two bytes of the hash for ShortMessageSign
        // 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
        // iterate to take bytes from SHA-256
        for (int i=0; i< bigDigest.length; i++){
            messageDigest[i+1] = bigDigest[i];
```

```
        }
        // 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();
    }

    /**
     * use this method to send and receive message from or to server
     * @param message
     * @throws IOException
     * @throws NoSuchAlgorithmException
     */
    public void communicate(String message) throws IOException, NoSuchAlgorithmExcepti
on {
        String signature = getSignature(message);
        String requestString = publicKey+","+message+","+signature;
        // send request to server
        BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getI
nputStream()));
        PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(cl
ientSocket.getOutputStream())));
        out.println(requestString);
        out.flush();
        // receive response from server
        String replyString = in.readLine();
        System.out.println("The result is "+replyString+".");
    }

    /**
     * create a new client TCP to do proxy design.
     * @param args
     */
    public static void main(String[] args)  {
        System.out.println("The client is running.");
        Scanner sc = new Scanner(System.in);
        System.out.println("Please enter server port: ");
        int portNum = sc.nextInt();
        try {
            SigningClientTCP clientTCP = new SigningClientTCP(portNum);
            String en = String.valueOf(clientTCP.e)+String.valueOf(clientTCP.n);
            // generate client ID
            clientTCP.setId(en);
            // display public and private keys
            System.out.println("Public Key: "+"("+clientTCP.publicKey+")");
            System.out.println("Private Key: "+"("+clientTCP.privateKey+")");

            while (true){
                String operationString = clientTCP.start(sc);
                if (operationString.equals("halt")){
                    break;
                }
                // sign each request and send
                clientTCP.communicate(operationString);
            }
        } catch (IOException e){
```

```
            System.out.println("IO Exception: "+e.getMessage());
        } catch (NoSuchAlgorithmException ex) {
            System.out.println("No such algorithm exception: "+ex.getMessage());;
        }

    }

}
```

- **Project2Task5Server**

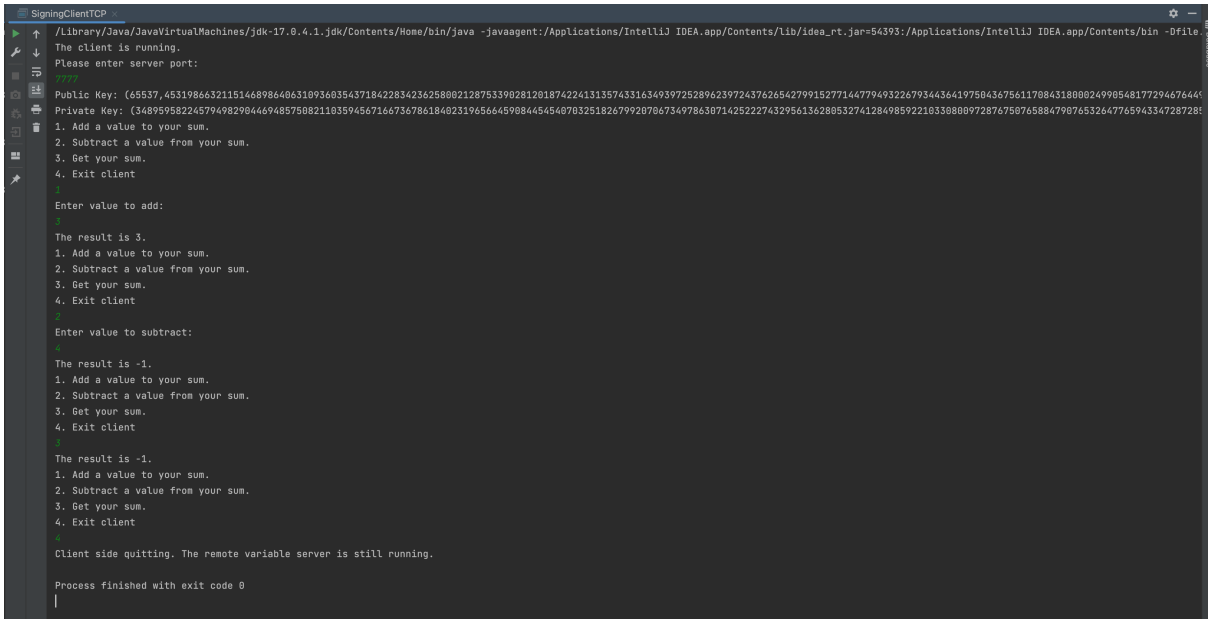
```
import java.net.*;
import java.io.*;
import java.util.HashMap;
import java.util.Scanner;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;
import java.util.TreeMap;

/**
 * Server side:
 * 1. two checks before servicing requests from client:
 *  - public key hash to ID
 *  - request properly signed
 *  - otherwise, return the message "Error in request"
 * 2. use SHA-256 for our hash function h()
 */

public class VerifyingServerTCP {
    private Socket clientSocket;
    private ServerSocket aSocket;
    private int sum, option;
    private String id;
    private TreeMap<String, Integer> hashMap;
    private String signature;
    // for public key
    private BigInteger e,n;
    public VerifyingServerTCP() throws IOException {
        aSocket = new ServerSocket(7777);
        hashMap = new TreeMap<>();
        sum = 0;
    }
    /**
     * check if id equals to public key hash
     * @return
     * @throws IOException
     * @throws NoSuchAlgorithmException
     */
    public boolean check() throws IOException, NoSuchAlgorithmException {
        String en = String.valueOf(e)+String.valueOf(n);
        if (id.equals(new SigningClientTCP(7777).setId(en))){
            System.out.println("Client public key: "+("+e+","+n+")");
            return true;
```

```
        }
        System.out.println("Error in request.");
        return false;
    }

    /**
     * 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 requests
     * @return whether the request is correctly encrypted.
     */
    public boolean decrypt(String [] requests) throws UnsupportedEncodingException, No
SuchAlgorithmException {
        // Take the encrypted string and make it a big integer
        BigInteger encryptedHash = new BigInteger(signature);
        // Decrypt it
        BigInteger decryptedHash = encryptedHash.modPow(e, n);
        // concat another operation string
        String messageToCheck = "";
        for (int i=2; i<requests.length-1; i++){
            messageToCheck += (requests[i]+",");
        }
        // Get the bytes from operation string
        byte[] bytesOfMessageToCheck = messageToCheck.substring(0,messageToCheck.lengt
h()-1).getBytes("UTF-8");
        // compute the digest of the message with SHA-256
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] messageToCheckDigest = md.digest(bytesOfMessageToCheck);
        // messageToCheckDigest is a full SHA-256 digest
        // take two bytes from SHA-256 and add a zero byte
        byte[] extraByte = new byte[messageToCheckDigest.length+1];
        extraByte[0] = 0;
        for (int i =0; i<messageToCheckDigest.length; i++){
            extraByte[i+1] = messageToCheckDigest[i];
        }
        // Make it a big int
        BigInteger bigIntegerToCheck = new BigInteger(extraByte);
        // inform the client on how the two compare
        if(bigIntegerToCheck.compareTo(decryptedHash) == 0) {
            System.out.println("Valid signature.");
            return true;
        } else {
            System.out.println("Error in request.");
            return false;
        }
    }

    /**
     * get requests from client and extract elements from requestString
     * @param requests
     */
    public void init(String[] requests){
        e = new BigInteger(requests[0]);
        n = new BigInteger(requests[1]);
        id = requests[2];
```

```
        option = Integer.valueOf(requests[3]);
        signature = requests[requests.length-1];
    }
    /**
     * add number for each client ID
     * @param id
     * @param value
     * @return the result being returned and printed out
     */
    public int addNum(String id, int value){
        if (hashMap.containsKey(id)){
            sum = hashMap.get(id)+value;
            hashMap.put(id, sum);
        } else {
            sum = value;
            hashMap.put(id, sum);
        }
        return sum;
    }

    /**
     * subtract number for each client ID
     * @param id
     * @param value
     * @return the result being returned and printed out
     */
    public int subNum(String id, int value){
        if (hashMap.containsKey(id)){
            sum = hashMap.get(id)-value;
            hashMap.put(id, sum);
        } else {
            sum = 0-value;
            hashMap.put(id, sum);
        }
        return sum;
    }
    public static void main(String[] args) {
        System.out.println("Server Started.");
        try {
            VerifyingServerTCP serverTCP = new VerifyingServerTCP();
            serverTCP.clientSocket = serverTCP.aSocket.accept();
            Scanner in = new Scanner(serverTCP.clientSocket.getInputStream());
            PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWrite
r(serverTCP.clientSocket.getOutputStream())));
            String requestString="";
            while (true){
                if (serverTCP.clientSocket==null){
                    serverTCP.clientSocket = serverTCP.aSocket.accept();
                    in = new Scanner(serverTCP.clientSocket.getInputStream());
                    out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(se
rverTCP.clientSocket.getOutputStream())));
                }
                try{
                    requestString = in.nextLine();
                } catch(Exception e){
                    serverTCP.clientSocket.close();
                    serverTCP.clientSocket = null;
                    continue;
```

```
                }
                //System.out.println(requestString);
                String[] requests = requestString.split(",");
                serverTCP.init(requests);
                // check public key hash to ID
                if (!serverTCP.check())
                    break;
                // check if request is properly signed
                if (!serverTCP.decrypt(requests))
                    break;
                // perform the operation
                int result = 0;
                int value = 0;

                switch (serverTCP.option){
                    case 1:
                        value = Integer.parseInt(requests[4]);
                        result = serverTCP.addNum(serverTCP.id, value);
                        break;
                    case 2:
                        value = Integer.parseInt(requests[4]);
                        result = serverTCP.subNum(serverTCP.id, value);
                        break;
                    case 3:
                        result = serverTCP.hashMap.get(serverTCP.id);
                        break;
                    default:
                        System.out.println("invalid option input! Will return 0 for th
is option.");

                }
                System.out.println("Client ID: "+serverTCP.id+"; Operation Request: "+
serverTCP.option+"; Returned Value: "+result);
                // send the result to the client
                out.println(result+"\n");
                out.flush();
            }

        } catch (IOException ex) {
            System.out.println("IO Exception: "+ex.getMessage());
        } catch (NoSuchAlgorithmException ex) {
            System.out.println("No such algorithm exception: "+ex.getMessage());;
        }

    }

}
```

- **Project2Task5ClientConsole**



- **Project2Task5ServerConsole**