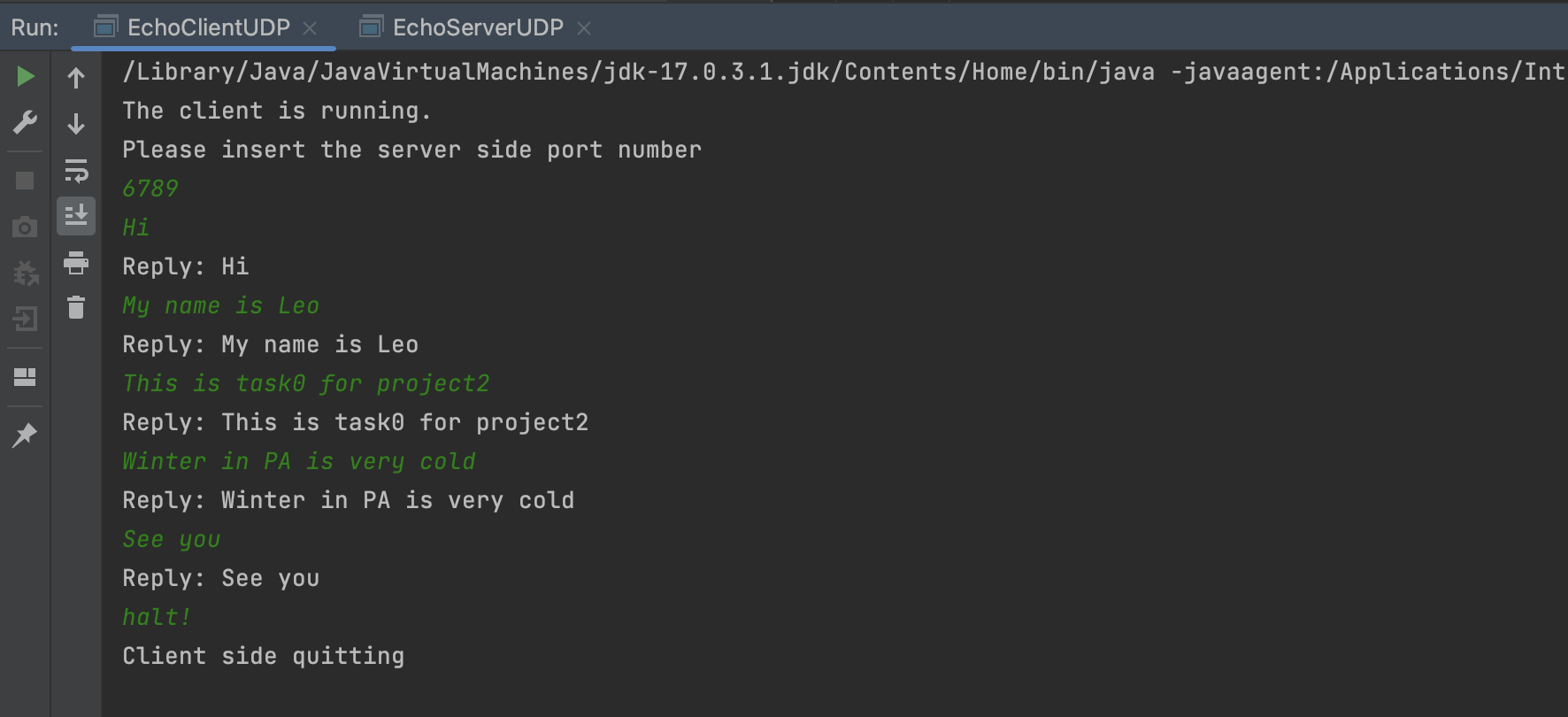
**Project2Task0Client**

import java.net.\*;  
import java.io.\*;  
import java.util.Scanner;  
// Name: Leo Lin  
// Andrew ID: hungfanl  
public class EchoClientUDP{  
 public static void main(String args[]){  
 // Showing that the client UDP is running  
 System.*out*.println("The client is running.");  
 // args give message contents and server hostname  
 DatagramSocket aSocket = null;  
 try {  
 // collecting IP address  
 InetAddress aHost = InetAddress.*getByName*("localhost");  
 // The client is using 6789 as its port number  
 Scanner readInput = new Scanner(System.*in*);  
 System.*out*.println("Please insert the server side port number");  
 int serverPort = readInput.nextInt();  
 // set up a new socket  
 aSocket = new DatagramSocket();  
 String nextLine;  
 // Initiate a BufferedReader that record the user's input  
 BufferedReader typed = new BufferedReader(new InputStreamReader(System.*in*));  
 // Transfer the user's input to byte form and transfer to the server  
 while ((nextLine = typed.readLine()) != null) {  
 // get byte form and turn into a DatagramPacket file  
 byte [] m = nextLine.getBytes();  
 DatagramPacket request = new DatagramPacket(m, m.length, aHost, serverPort);  
 // send the request  
 aSocket.send(request);  
  
 byte[] buffer = new byte[1000];  
 DatagramPacket reply = new DatagramPacket(buffer, buffer.length);  
 // receive the reply from the server side and print it out.  
 aSocket.receive(reply);  
 String replyString = new String(reply.getData()).substring(0, reply.getLength());  
 // If replyString is "halt!" break the while loop  
 if(replyString.equals("halt!")) {  
 System.*out*.println("Client side quitting");  
 break;  
 }  
 System.*out*.println("Reply: " + replyString);  
 }  
  
 }catch (SocketException e) {System.*out*.println("Socket: " + e.getMessage());  
 }catch (IOException e){System.*out*.println("IO: " + e.getMessage());  
 }finally {  
 if(aSocket != null) aSocket.close();  
 }  
 }  
}

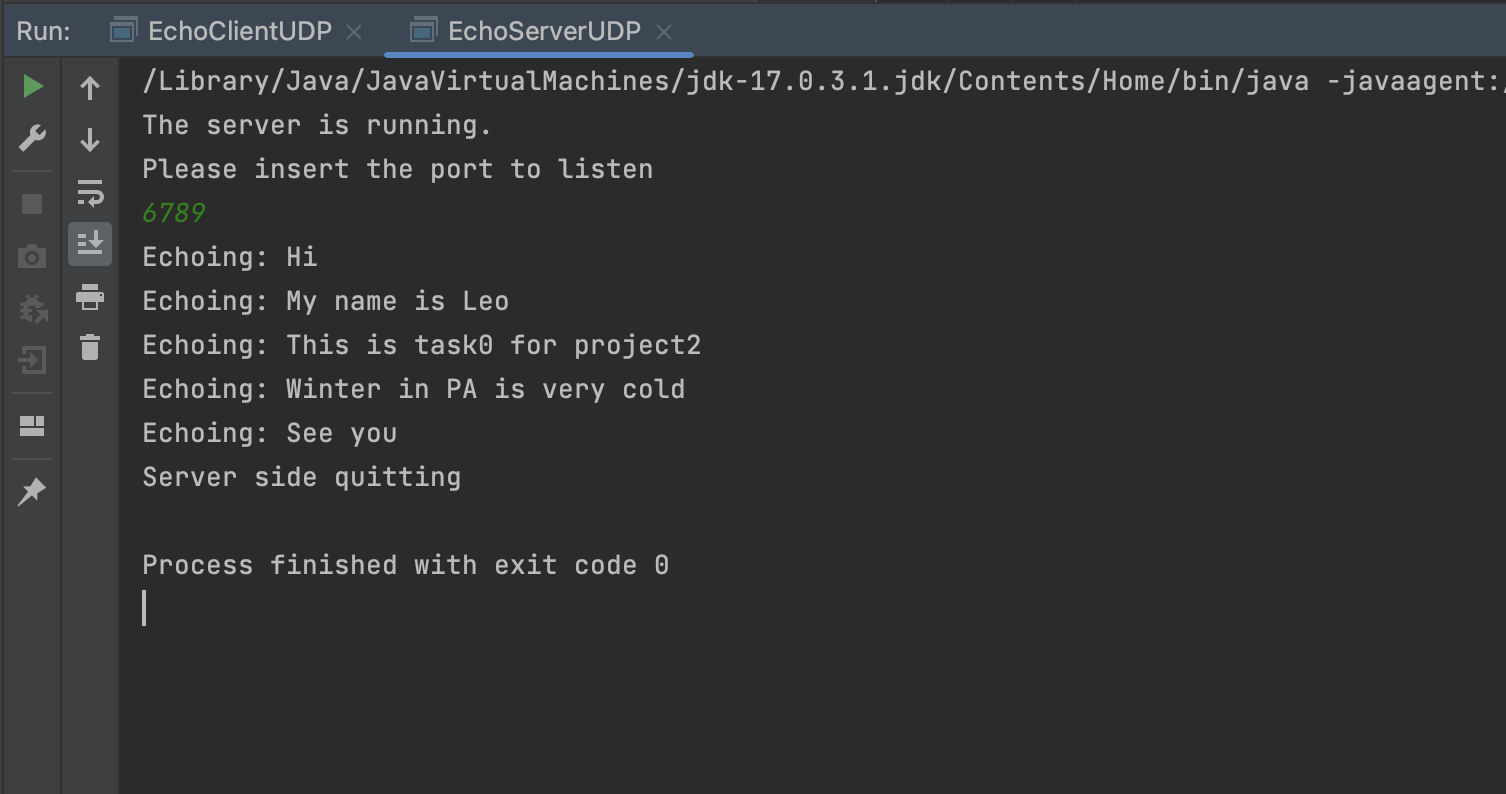
**Project2Task0Server**

import java.net.\*;  
import java.io.\*;  
import java.util.Scanner;  
// Name: Leo Lin  
// Andrew ID: hungfanl  
  
public class EchoServerUDP{  
 public static void main(String args[]){  
 DatagramSocket aSocket = null;  
 byte[] buffer = new byte[1000];  
 System.*out*.println("The server is running.");  
 Scanner readInput = new Scanner(System.*in*);  
 System.*out*.println("Please insert the port to listen");  
 int listenPort = readInput.nextInt();  
 try{  
 aSocket = new DatagramSocket(listenPort);  
 DatagramPacket request = new DatagramPacket(buffer, buffer.length);  
 // Whenever the aSocket receive a request from the client side, it will document its data, length, address  
 // and port into "reply". It will then print out the request string and send a reply back  
 while(true){  
 // receive a request from the client  
 aSocket.receive(request);  
 // Create a DatagramPacket object with the request  
 DatagramPacket reply = new DatagramPacket(request.getData(),  
 request.getLength(), request.getAddress(), request.getPort());  
 String requestString = new String(request.getData()).substring(0,request.getLength());  
 // send the reply back to the client  
 aSocket.send(reply);  
 if(requestString.equals("halt!")){  
 System.*out*.println("Server side quitting");  
 break;  
 }  
 // print out the request in a string form from byte form  
 System.*out*.println("Echoing: "+requestString);  
 }  
 }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**



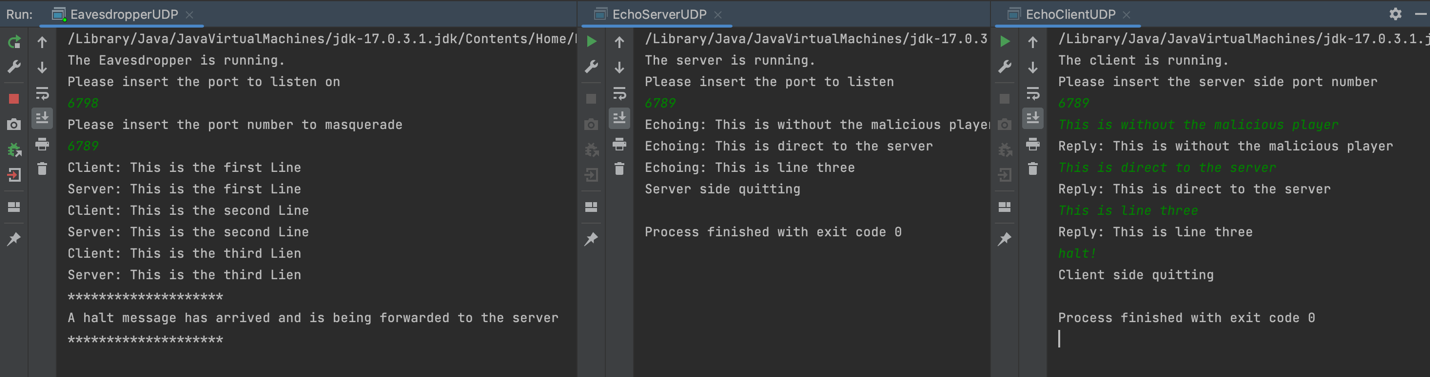
**Project2Task0ServerConsole**

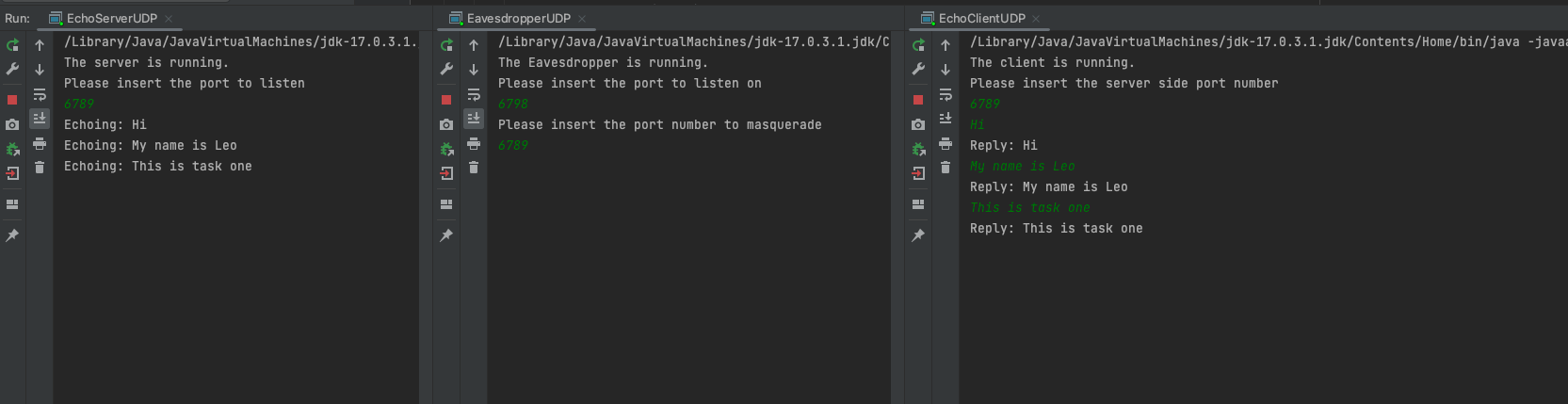


**Task 1 EavesdropperUDP**

import java.net.\*;  
import java.io.\*;  
import java.util.Scanner;  
// Name: Leo Lin  
// Andrew ID: hungfanl  
public class EchoClientUDP{  
 public static void main(String args[]){  
 // Showing that the client UDP is running  
 System.*out*.println("The client is running.");  
 // args give message contents and server hostname  
 DatagramSocket aSocket = null;  
 try {  
 // collecting IP address  
 InetAddress aHost = InetAddress.*getByName*("localhost");  
 // The client is using 6789 as its port number  
 Scanner readInput = new Scanner(System.*in*);  
 System.*out*.println("Please insert the server side port number");  
 int serverPort = readInput.nextInt();  
 // set up a new socket  
 aSocket = new DatagramSocket();  
 String nextLine;  
 // Initiate a BufferedReader that record the user's input  
 BufferedReader typed = new BufferedReader(new InputStreamReader(System.*in*));  
 // Transfer the user's input to byte form and transfer to the server  
 while ((nextLine = typed.readLine()) != null) {  
 // get byte form and turn into a DatagramPacket file  
 byte [] m = nextLine.getBytes();  
 DatagramPacket request = new DatagramPacket(m, m.length, aHost, serverPort);  
 // send the request  
 aSocket.send(request);  
  
 byte[] buffer = new byte[1000];  
 DatagramPacket reply = new DatagramPacket(buffer, buffer.length);  
 // receive the reply from the server side and print it out.  
 aSocket.receive(reply);  
 String replyString = new String(reply.getData()).substring(0, reply.getLength());  
 // If replyString is "halt!" break the while loop  
 if(replyString.equals("halt!")) {  
 System.*out*.println("Client side quitting");  
 break;  
 }  
 System.*out*.println("Reply: " + replyString);  
 }  
  
 }catch (SocketException e) {System.*out*.println("Socket: " + e.getMessage());  
 }catch (IOException e){System.*out*.println("IO: " + e.getMessage());  
 }finally {  
 if(aSocket != null) aSocket.close();  
 }  
 }  
}

**Project2Task1ThreeConsoles**



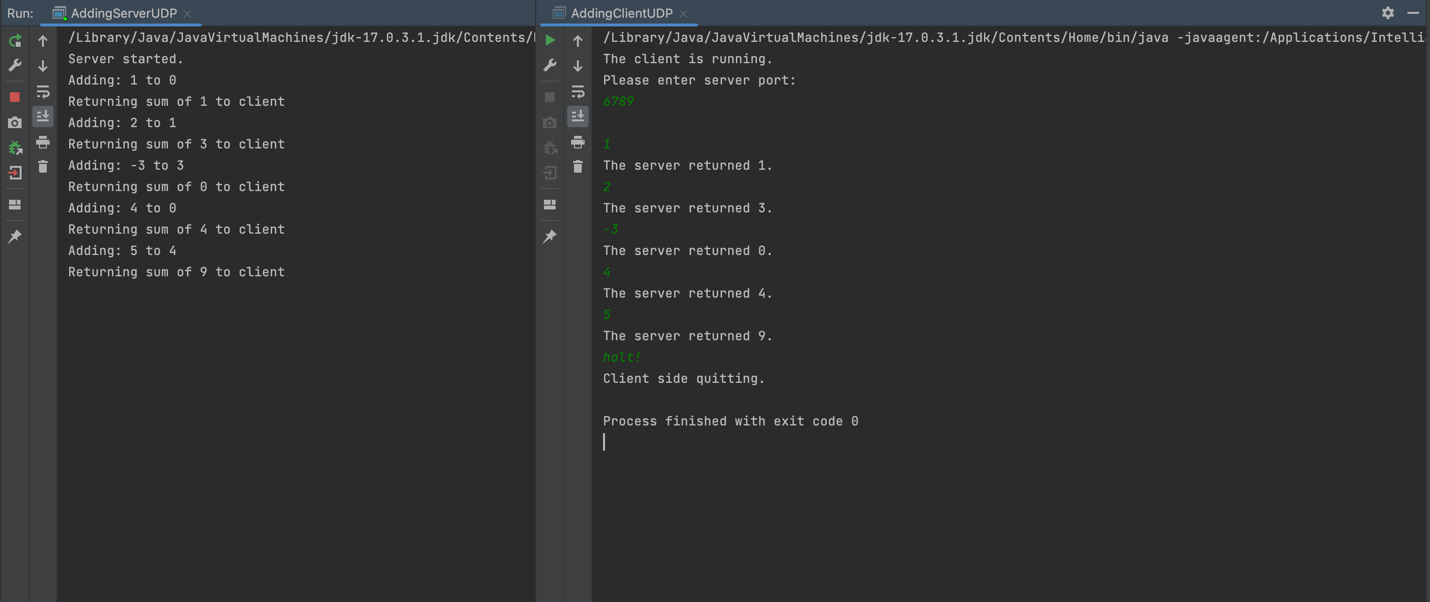


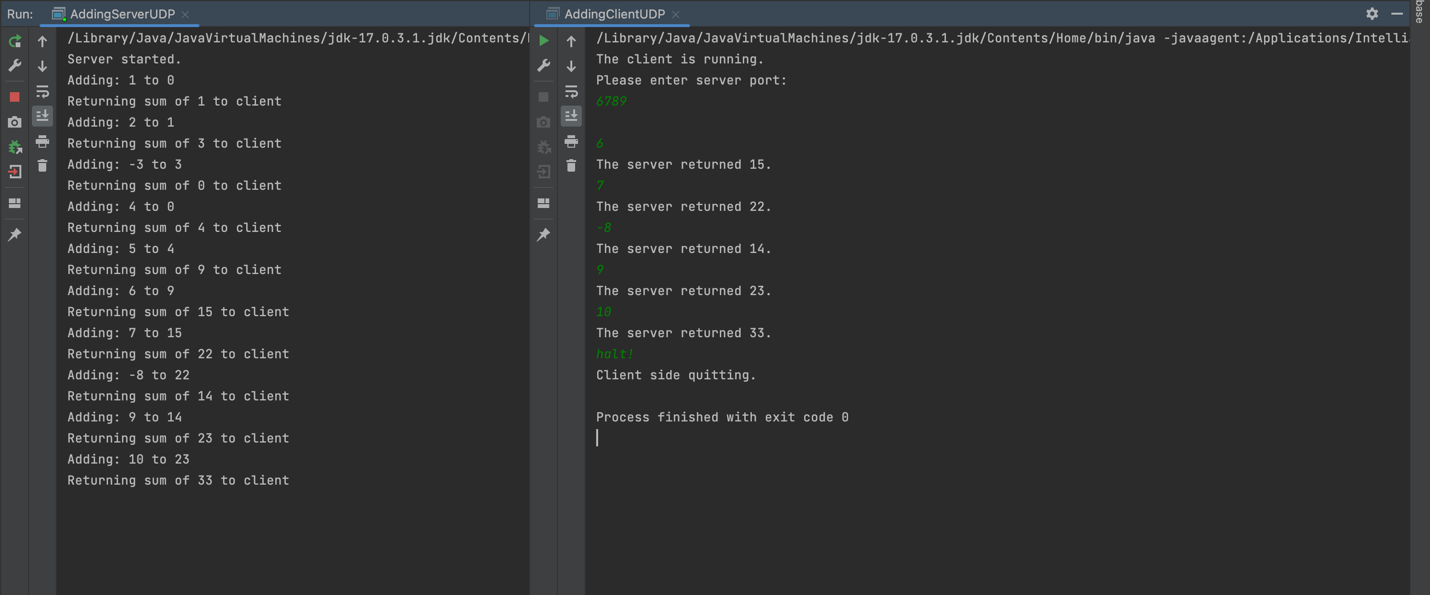
**Project2Task2Client**

import java.net.\*;  
import java.io.\*;  
import java.util.Scanner;  
// Name: Leo Lin  
// Andrew ID: hungfanl  
public class AddingClientUDP{  
 static DatagramSocket *aSocket* = null;  
 static int *serverPort*;  
 static InetAddress *aHost*;  
 public static void main(String args[]){  
 // Showing that the client UDP is running  
 Scanner readInput = new Scanner(System.*in*);  
 System.*out*.println("The client is running.");  
 // get the port number from the client  
 System.*out*.println("Please enter server port:");  
 *serverPort* = Integer.*parseInt*(readInput.nextLine());  
 try {  
 // collecting IP address  
 *aHost* = InetAddress.*getByName*("localhost");  
 *aSocket* = new DatagramSocket();  
 }catch (SocketException e) {System.*out*.println("Socket: " + e.getMessage());  
 }catch (IOException e){System.*out*.println("IO: " + e.getMessage());  
 }  
 String s;  
 System.*out*.println();  
 do{  
 s = readInput.nextLine();  
 // If the client does not input "halt!" initiate the add function  
 if(!s.equals("halt!")){  
 int result = *add*(Integer.*parseInt*(s));  
 System.*out*.println("The server returned " + result +".");  
 }  
 }while(!s.equals("halt!"));  
  
 System.*out*.println("Client side quitting.");  
 if(*aSocket* != null) *aSocket*.close();  
  
  
 }  
  
 //This function takes a integer from the user, pass it to the server, and return the respond of the server  
 public static int add(int i){  
 String replyString = null;  
 try {  
 byte [] m = String.*valueOf*(i).getBytes();  
 DatagramPacket request = new DatagramPacket(m, m.length, *aHost*, *serverPort*);  
 // send the request  
 *aSocket*.send(request);  
 byte[] buffer = new byte[1000];  
 DatagramPacket reply = new DatagramPacket(buffer, buffer.length);  
 // receive the reply from the server side and print it out.  
 *aSocket*.receive(reply);  
 replyString = new String(reply.getData()).substring(0, reply.getLength());  
 // If replyString is "halt!" break the while loop  
 }catch (SocketException e) {System.*out*.println("Socket: " + e.getMessage());  
 }catch (IOException e){System.*out*.println("IO: " + e.getMessage());  
 }  
 return Integer.*parseInt*(replyString);  
 }  
}

**Project2Task2Server**

import java.net.\*;  
import java.io.\*;  
import java.util.Scanner;  
// Name: Leo Lin  
// Andrew ID: hungfanl  
  
public class AddingServerUDP{  
 static int *sum*;  
 public static void main(String args[]){  
 *sum* = 0;  
 DatagramSocket aSocket = null;  
 byte[] buffer = new byte[1000];  
 System.*out*.println("Server started.");  
 int listenPort = 6789;  
 try{  
 aSocket = new DatagramSocket(listenPort);  
 DatagramPacket request = new DatagramPacket(buffer, buffer.length);  
 // Whenever the aSocket receive a request from the client side, it will add it to the sum variable  
 // and return the sum value. It will then print out the sum  
 while(true){  
 // receive a request from the client  
 aSocket.receive(request);  
 // Create a DatagramPacket object with the request  
 String requestString = new String(request.getData()).substring(0,request.getLength());  
 // add the integer from the client and add it to the sum variable  
 *add*(Integer.*parseInt*(requestString));  
 byte [] m = String.*valueOf*(*sum*).getBytes();  
 DatagramPacket reply = new DatagramPacket(m,  
 m.length, request.getAddress(), request.getPort());  
  
 // send the reply back to the client  
 System.*out*.println("Returning sum of " + *sum* + " to client");  
 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();}  
 }  
  
 public static void add(int number){  
 System.*out*.println("Adding: " + number + " to " + *sum*);  
 *sum* += number;  
 }  
}

**Project2Task2ClientConsole + Project2Task2ServerConsole**

****

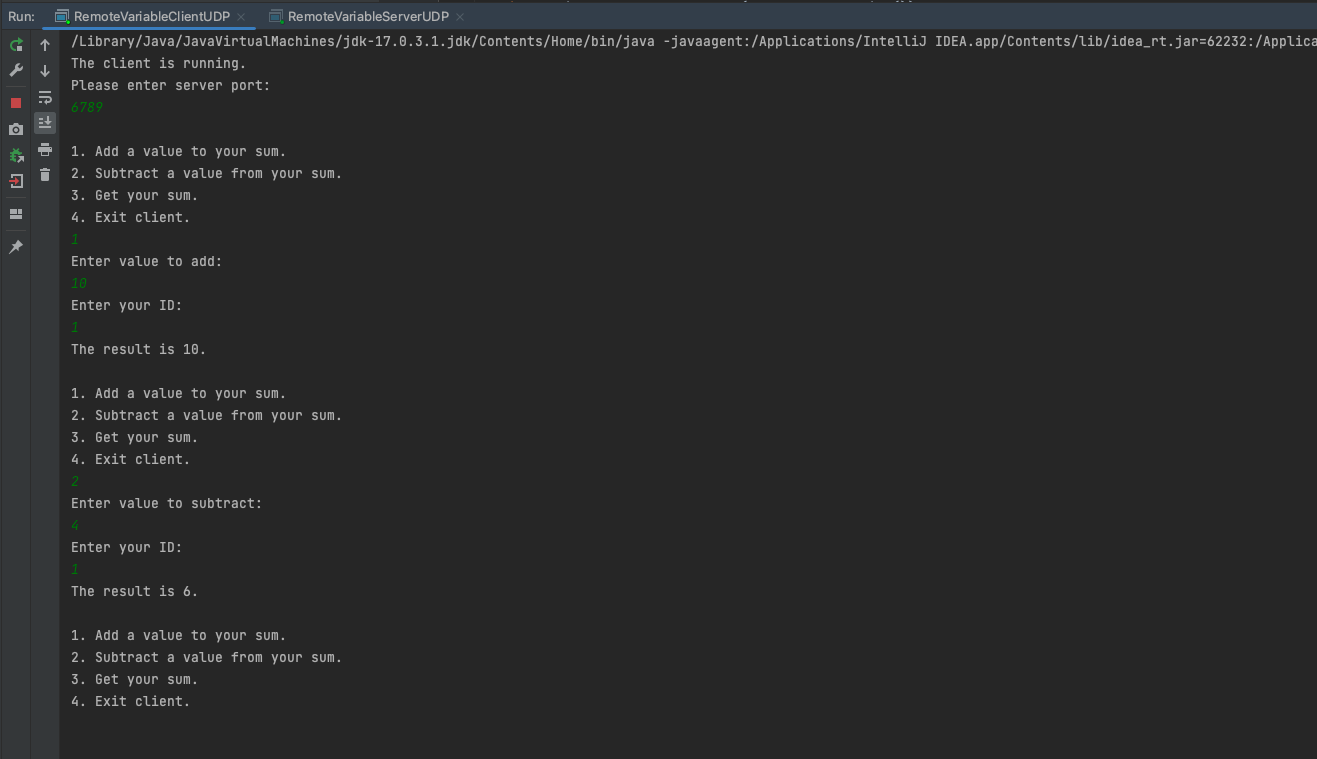
**Project2Task3Client**

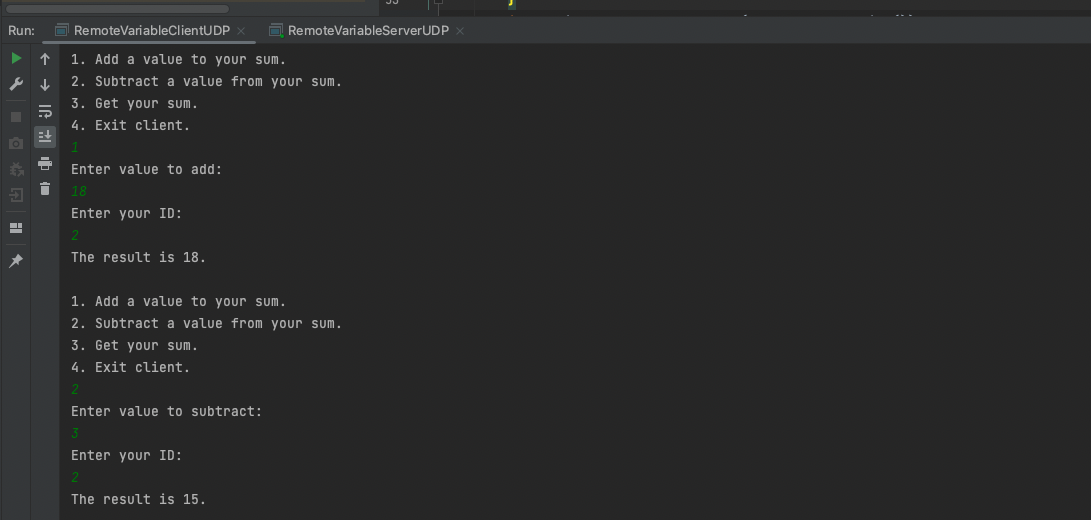
import java.net.\*;  
import java.io.\*;  
import java.util.Scanner;  
  
// Name: Leo Lin  
// Andrew ID: hungfanl  
public class RemoteVariableClientUDP{  
 static DatagramSocket *aSocket* = null;  
 static int *serverPort*;  
 static InetAddress *aHost*;  
 public static void main(String args[]){  
 // Showing that the client UDP is running  
 Scanner readInput = new Scanner(System.*in*);  
 System.*out*.println("The client is running.");  
 // get the port number from the client  
 System.*out*.println("Please enter server port:");  
 *serverPort* = Integer.*parseInt*(readInput.nextLine());  
 try {  
 // collecting IP address  
 *aHost* = InetAddress.*getByName*("localhost");  
 *aSocket* = new DatagramSocket();  
 }catch (SocketException e) {System.*out*.println("Socket: " + e.getMessage());  
 }catch (IOException e){System.*out*.println("IO: " + e.getMessage());  
 }  
 String option;  
 System.*out*.println();  
 // Keep on looping until the user insert option 4  
 do{  
 // Initiate the getOption function  
 option = *getOption*();  
 // If user doesn't insert 4, then pass function will pass the string to the server  
 if(!option.equals("4")){  
 int result = *pass*(option);  
 System.*out*.println("The result is " + result +".");  
 System.*out*.println();  
 }  
 }while(!option.equals("4"));  
  
 System.*out*.println("Client side quitting. The remote variable server is still running.");  
 if(*aSocket* != null) *aSocket*.close();  
  
  
 }  
  
 // This function returns a string concatenating type, id, number  
 // And returns 4 if the user insert 4.  
 public static String getOption(){  
 Scanner readInput = new Scanner(System.*in*);  
 String[] options = {"Add a value to your sum.","Subtract a value from your sum.","Get your sum.","Exit client."};  
 for(int i = 0; i < options.length; i++){  
 System.*out*.println((i+1) + ". " + options[i]);  
 }  
 int choice = Integer.*parseInt*(readInput.nextLine());  
 String number = "0";  
 String id;  
 // return string based on user's insertion.  
 switch(choice) {  
 // if the user insert 4, then return 4 to exit the client program  
 case 4:  
 return String.*valueOf*(choice);  
 case 1:  
 System.*out*.println("Enter value to add:");  
 number = readInput.nextLine();  
 break;  
 case 2:  
 System.*out*.println("Enter value to subtract:");  
 number = readInput.nextLine();  
 break;  
 default:  
 }  
 System.*out*.println("Enter your ID:");  
 id = readInput.nextLine();  
 return String.*valueOf*(choice) + "," + id + "," + number;  
  
 }  
 //This function takes the concatenated string from the client and pass it to the server  
 public static int pass(String s){  
 String replyString = null;  
 try {  
 byte [] m = String.*valueOf*(s).getBytes();  
 DatagramPacket request = new DatagramPacket(m, m.length, *aHost*, *serverPort*);  
 // send the request  
 *aSocket*.send(request);  
 byte[] buffer = new byte[1000];  
 DatagramPacket reply = new DatagramPacket(buffer, buffer.length);  
 // receive the reply from the server side and print it out.  
 *aSocket*.receive(reply);  
 replyString = new String(reply.getData()).substring(0, reply.getLength());  
 // If replyString is "halt!" break the while loop  
 }catch (SocketException e) {System.*out*.println("Socket: " + e.getMessage());  
 }catch (IOException e){System.*out*.println("IO: " + e.getMessage());  
 }  
 return Integer.*parseInt*(replyString);  
 }  
}

**Project2Task3Server**

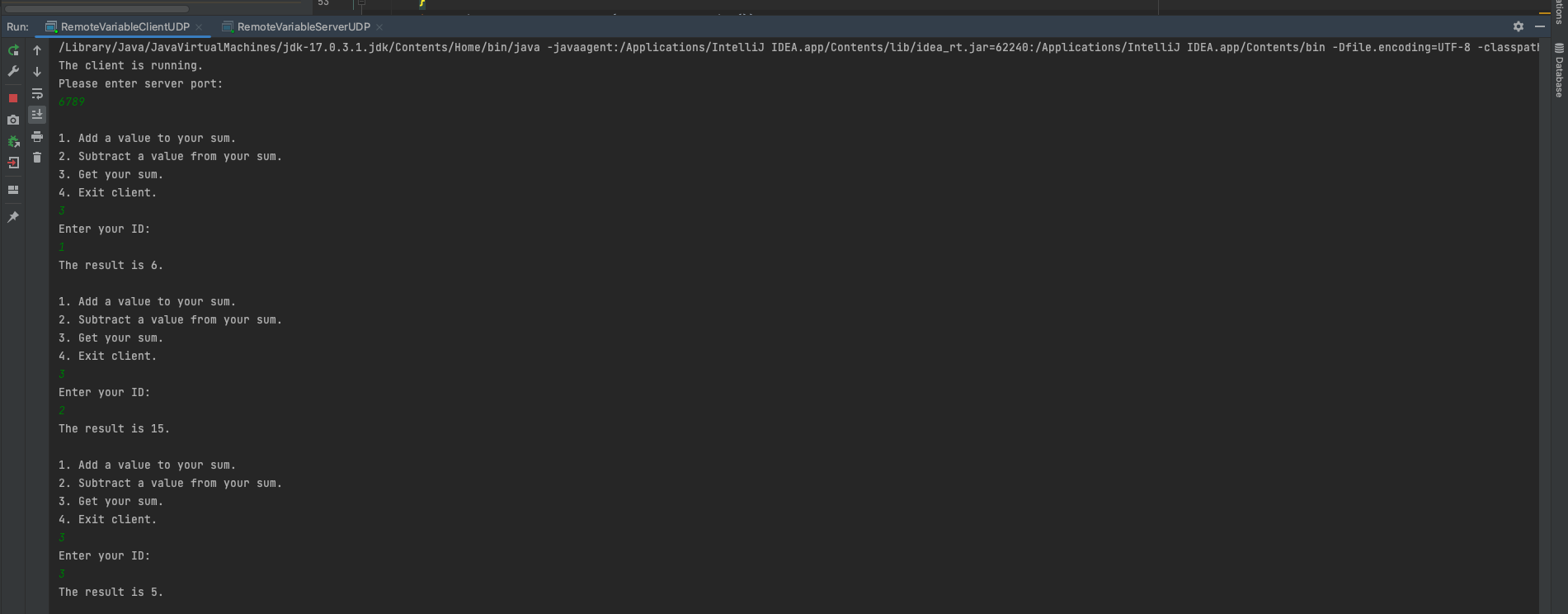
import java.net.\*;  
 import java.io.\*;  
import java.util.Map;  
import java.util.Scanner;  
import java.util.TreeMap;  
// Name: Leo Lin  
// Andrew ID: hungfanl  
  
public class RemoteVariableServerUDP{  
  
 static Map<Integer, Integer> *database* = new TreeMap<>();  
 public static void main(String args[]){  
 DatagramSocket aSocket = null;  
 byte[] buffer = new byte[1000];  
 System.*out*.println("Server started.");  
 int listenPort = 6789;  
 try{  
 aSocket = new DatagramSocket(listenPort);  
 DatagramPacket request = new DatagramPacket(buffer, buffer.length);  
 // Whenever the aSocket receive a request from the client side, it will add it to the sum variable  
 // and return the sum value. It will then print out the sum  
 while(true){  
 // receive a request from the client  
 aSocket.receive(request);  
 // create a DatagramPacket object with the request  
 String requestString = new String(request.getData()).substring(0,request.getLength());  
 // split the passed string into operator, id, and number.  
 String[] operation = requestString.split(",");  
 int operator = Integer.*parseInt*(operation[0]);  
 int id = Integer.*parseInt*(operation[1]);  
 int number = Integer.*parseInt*(operation[2]);  
 // conduct corresponding arithmetic according to the three index  
 int outcome = *arithmetic*(operator, id, number);  
 // return the outcome of calculation to the client  
 byte [] m = String.*valueOf*(outcome).getBytes();  
 DatagramPacket reply = new DatagramPacket(m,  
 m.length, request.getAddress(), request.getPort());  
 // send the reply back to the client  
 System.*out*.println("Returning sum of " + outcome + " to client");  
 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();}  
 }  
  
  
 // Doing corresponding arithmetic according to the user's request  
 public static int arithmetic(int operator, int id, int number){  
 String[] operation = {"Addition", "Subtraction", "Check"};  
 System.*out*.println("The visitor's ID is: " + id);  
 System.*out*.println("Operand: " + operation[operator-1]);  
 if(operator == 1){  
 *database*.put(id, *database*.getOrDefault(id,0) + number);  
 }  
 else if (operator == 2){  
 *database*.put(id, *database*.getOrDefault(id,0) - number);  
 }  
 // return the number after calculation  
 return *database*.get(id);  
 }  
}

**Project2Task3ClientConsole**

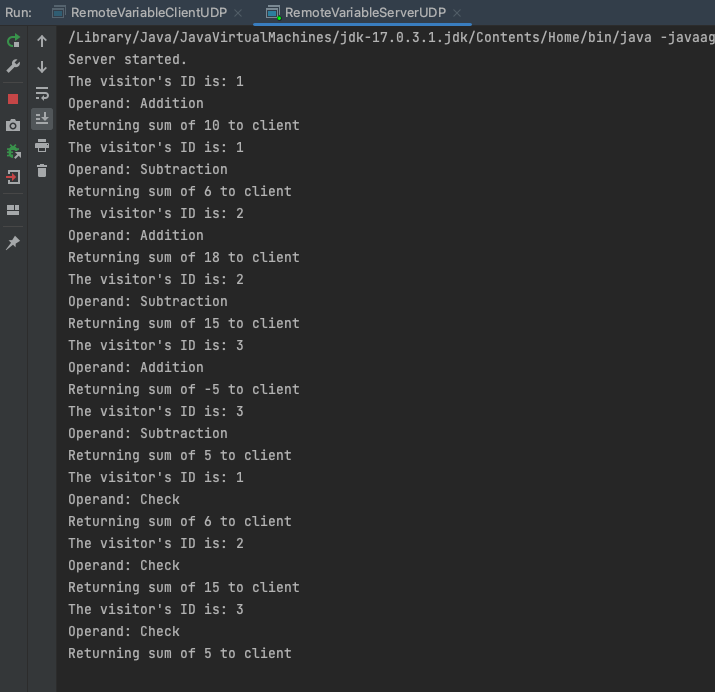
****

****





**Project2Task3ServerConsole**

****

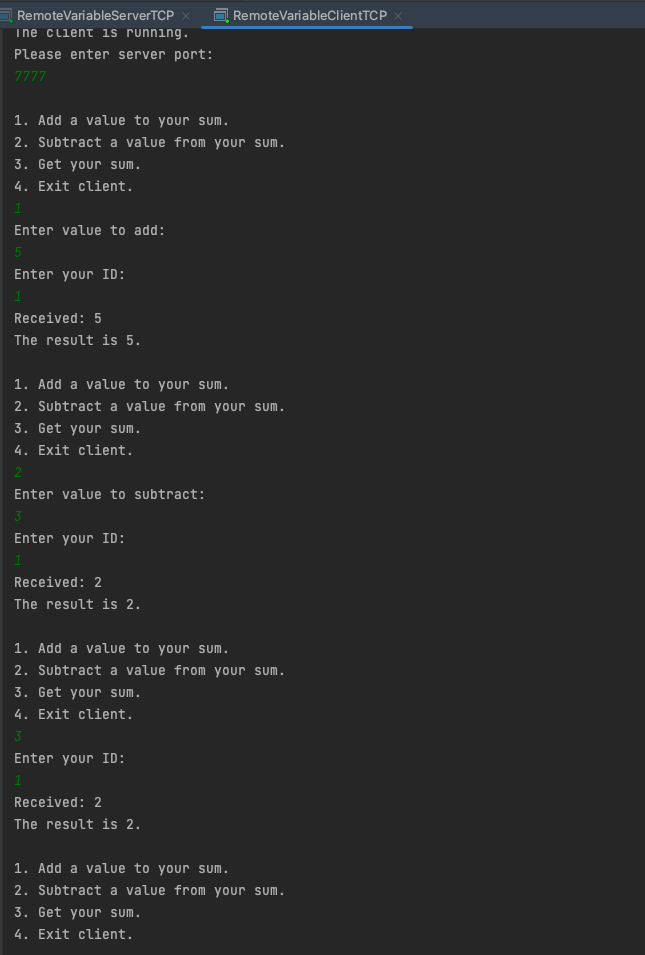
**Project2Task4Client**

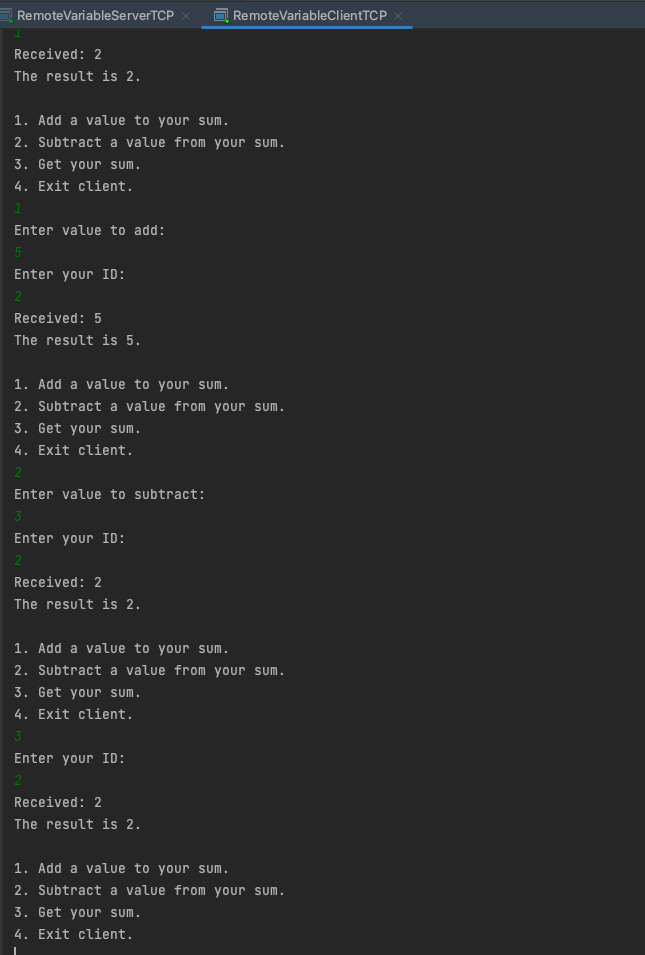
import java.net.\*;  
import java.io.\*;  
import java.util.Scanner;  
// Name: Leo Lin  
// Andrew ID: hungfanl  
public class RemoteVariableClientTCP {  
 static Socket *clientSocket* = null;  
 static int *serverPort*;  
 static BufferedReader *in*;  
 static PrintWriter *out*;  
 public static void main(String args[]) {  
 Scanner readInput = new Scanner(System.*in*);  
 // arguments supply hostname  
 System.*out*.println("The client is running.");  
 // get the port number from the client  
 System.*out*.println("Please enter server port:");  
 *serverPort* = readInput.nextInt();  
 try {  
 *clientSocket* = new Socket("localhost", *serverPort*);  
 BufferedReader typed = new BufferedReader(new InputStreamReader(System.*in*));  
 String option;  
 System.*out*.println();  
 do{  
 // Initiate the getOption function  
 option = *getOption*();  
 // If user doesn't insert 4, then pass function will pass the string to the server  
 if(!option.equals("4")){  
 int result = *pass*(option);  
 System.*out*.println("The result is " + result +".");  
 System.*out*.println();  
 }  
 }while(!option.equals("4"));  
 System.*out*.println("Client side quitting. The remote variable server is still running.");  
 } catch (IOException e) {  
 System.*out*.println("IO Exception:" + e.getMessage());  
 } finally {  
 try {  
 if (*clientSocket* != null) {  
 *clientSocket*.close();  
 }  
 } catch (IOException e) {  
 // ignore exception on close  
 }  
 }  
 }  
 // This function returns a string concatenating type, id, number  
 // And returns 4 if the user insert 4.  
 public static String getOption(){  
 Scanner readInput = new Scanner(System.*in*);  
 String[] options = {"Add a value to your sum.","Subtract a value from your sum.","Get your sum.","Exit client."};  
 for(int i = 0; i < options.length; i++){  
 System.*out*.println((i+1) + ". " + options[i]);  
 }  
 int choice = Integer.*parseInt*(readInput.nextLine());  
 String number = "0";  
 String id;  
 // return string based on user's insertion.  
 switch(choice) {  
 // if the user insert 4, then return 4 to exit the client program  
 case 4:  
 return String.*valueOf*(choice);  
 case 1:  
 System.*out*.println("Enter value to add:");  
 number = readInput.nextLine();  
 break;  
 case 2:  
 System.*out*.println("Enter value to subtract:");  
 number = readInput.nextLine();  
 break;  
 default:  
 }  
 System.*out*.println("Enter your ID:");  
 id = readInput.nextLine();  
 return String.*valueOf*(choice) + "," + id + "," + number;  
 }  
 //This function takes the concatenated string from the client and pass it to the server  
 public static int pass(String s){  
 String data = null;  
 try {  
 *in* = new BufferedReader(new InputStreamReader(*clientSocket*.getInputStream()));  
 *out* = new PrintWriter(new BufferedWriter(new OutputStreamWriter(*clientSocket*.getOutputStream())));  
 *out*.println(s);  
 *out*.flush();  
 data = *in*.readLine(); // read a line of data from the stream  
 System.*out*.println("Received: " + data);  
 } catch (IOException e) {  
 System.*out*.println("IO Exception:" + e.getMessage());  
 }  
 return Integer.*parseInt*(data);  
 }  
}

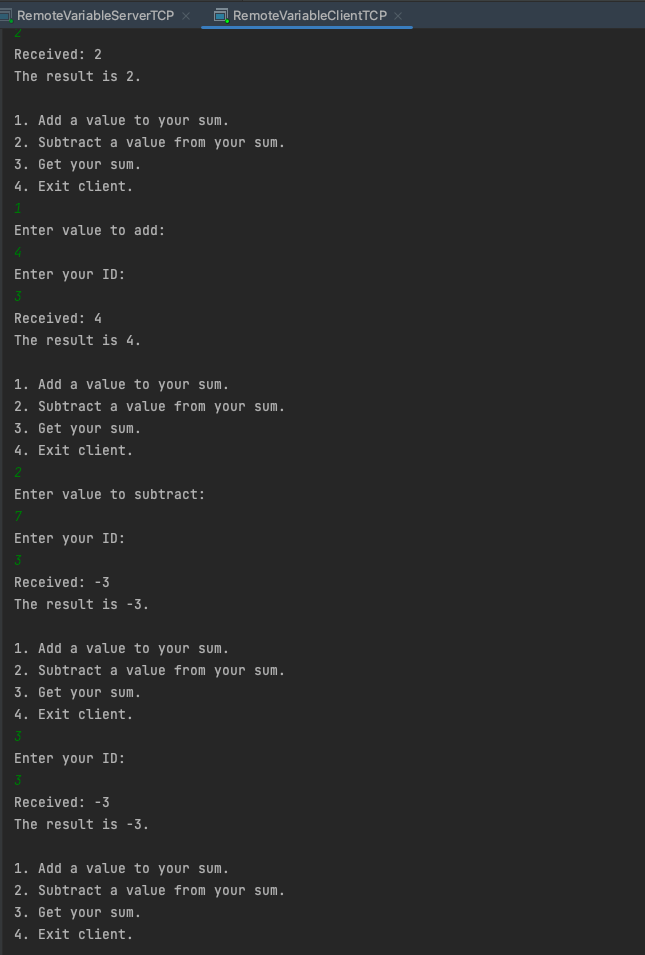
**Project2Task4Server**

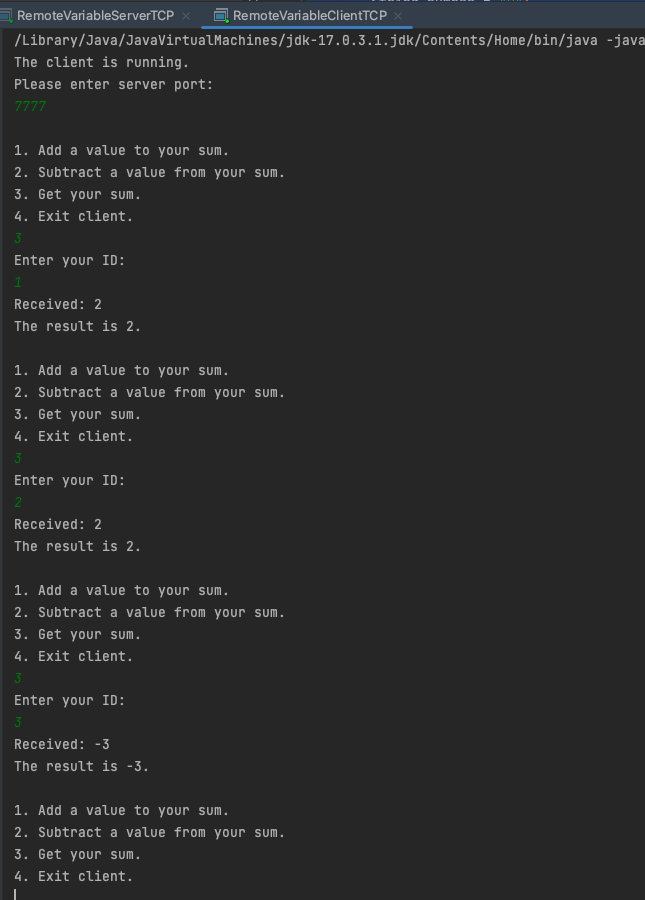
import java.net.\*;  
import java.io.\*;  
import java.util.Map;  
import java.util.Scanner;  
import java.util.TreeMap;  
// Name: Leo Lin  
// Andrew ID: hungfanl  
public class RemoteVariableServerTCP {  
 static Map<Integer, Integer> *database* = new TreeMap<>();  
 public static void main(String args[]) {  
 Socket clientSocket = null;  
 try {  
 int serverPort = 7777; // the server port we are using  
 // Create a new server socket  
 ServerSocket listenSocket = new ServerSocket(serverPort);  
 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())));  
 // In order for the server to run forever, we need to handle the situation when client shut down  
 while (true) {  
 String data;  
 // When the client is connected, there will be string pass to in.  
 // Under the situation, we can safely conduct the calculation and return the value.  
 if(in.hasNextLine()){  
 data = in.nextLine();  
 String[] operation = data.split(",");  
 int operator = Integer.*parseInt*(operation[0]);  
 int id = Integer.*parseInt*(operation[1]);  
 int number = Integer.*parseInt*(operation[2]);  
 // conduct corresponding arithmetic according to the three index  
 int outcome = *arithmetic*(operator, id, number);  
 // return the outcome of calculation to the client  
 System.*out*.println("Returning sum of " + outcome + " to client");  
 out.println(outcome);  
 out.flush();  
 }  
 // However, when the client is shut down, there will not be next line pass to in.  
 // In such cases, we will need to have clientSocket to accept another socket and renew the in/out  
 else {  
 clientSocket = listenSocket.accept();  
 in = new Scanner(clientSocket.getInputStream());  
 out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));  
 }  
 }  
 // Handle exceptions  
 } catch (IOException e) {  
 System.*out*.println("IO Exception:" + e.getMessage());  
  
 // If quitting (typically by you sending quit signal) clean up sockets  
 } finally {  
 try {  
 if (clientSocket != null) {  
 clientSocket.close();  
 }  
 } catch (IOException e) {  
 // ignore exception on close  
 }  
 }  
 }  
 // Doing corresponding arithmetic according to the user's request  
 public static int arithmetic(int operator, int id, int number){  
 String[] operation = {"Addition", "Subtraction", "Check"};  
 System.*out*.println("The visitor's ID is: " + id);  
 System.*out*.println("Operator: " + operation[operator-1]);  
 if(operator == 1){  
 *database*.put(id, *database*.getOrDefault(id,0) + number);  
 }  
 else if (operator == 2){  
 *database*.put(id, *database*.getOrDefault(id,0) - number);  
 }  
 // return the number after calculation  
 return *database*.get(id);  
 }  
}

**Project2Task4ClientConsole**

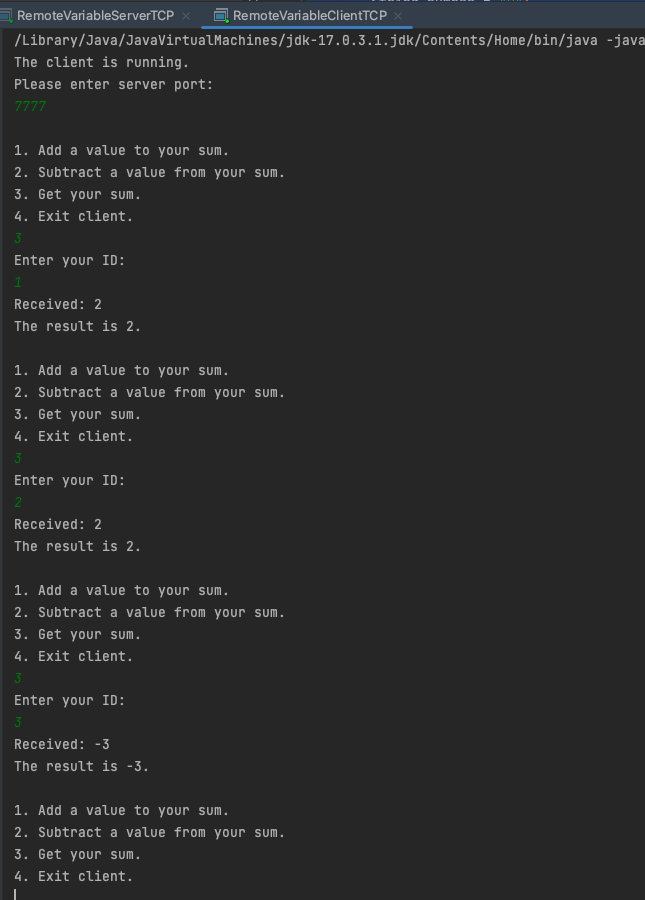




****

****

**Project2Task4ServerConsole**



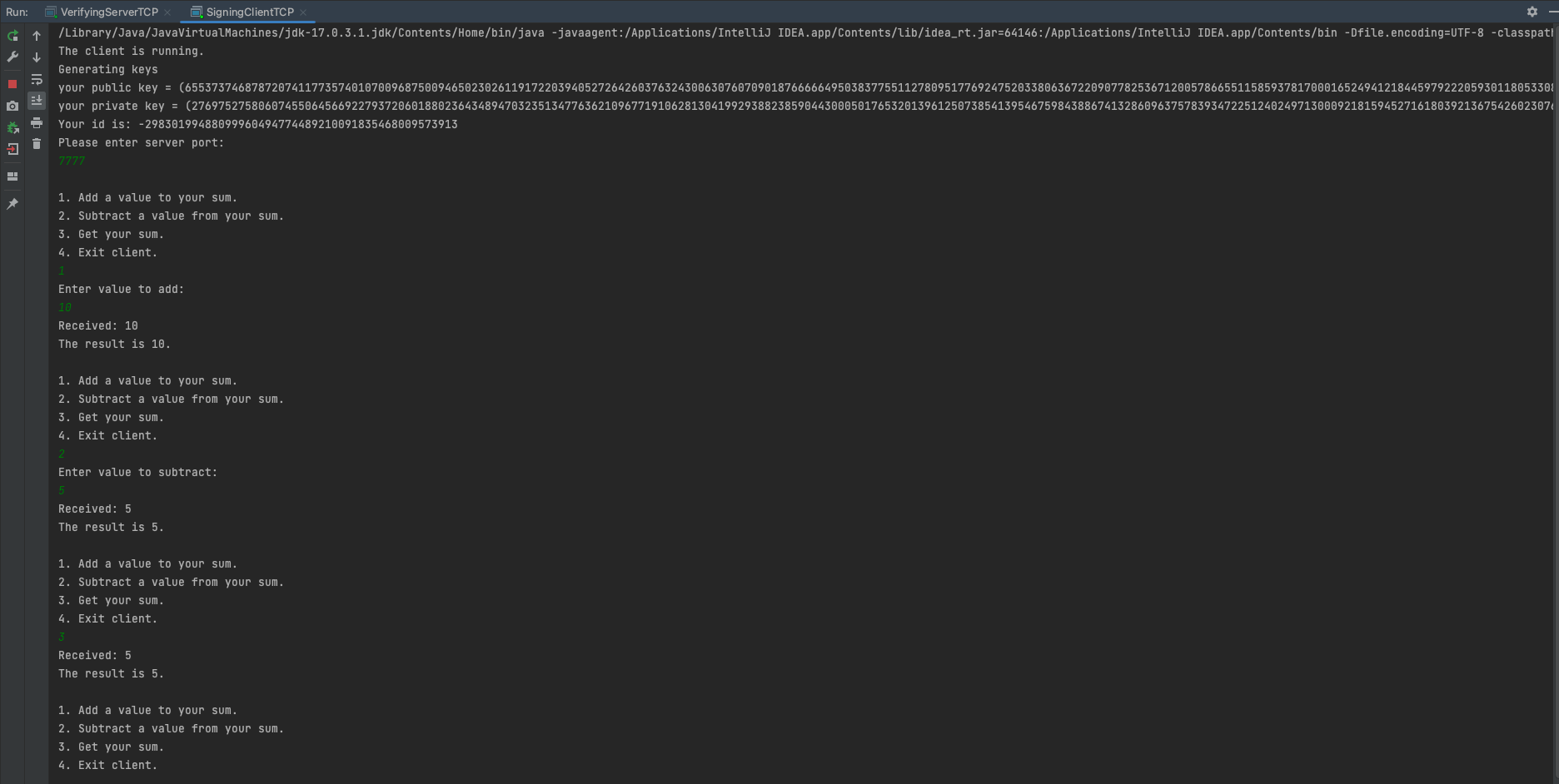
**Project2Task5Client**

import java.math.BigInteger;  
import java.net.\*;  
import java.io.\*;  
import java.security.NoSuchAlgorithmException;  
import java.util.Random;  
import java.util.Scanner;  
import java.security.MessageDigest;  
// Name: Leo Lin  
// Andrew ID: hungfanl  
public class SigningClientTCP {  
 static Socket *clientSocket* = null;  
 static int *serverPort*;  
 static BufferedReader *in*;  
 static PrintWriter *out*;  
 static BigInteger *n*, *e*, *d*, *id*;  
 static String *public\_key*, *private\_key*;  
 static final int *NUMBER\_OF\_ID\_DIGITS* = 20; // This is the  
 public static void main(String args[]) {  
 Scanner readInput = new Scanner(System.*in*);  
 // arguments supply hostname  
 System.*out*.println("The client is running.");  
 *generateKey*();  
 *generateID*();  
 // get the port number from the client  
 System.*out*.println("Please enter server port:");  
 *serverPort* = readInput.nextInt();  
 try {  
 *clientSocket* = new Socket("localhost", *serverPort*);  
 String message;  
 String encryptedMessage;  
 System.*out*.println();  
 do{  
 // Initiate the getOption function  
 message = *getMessage*();  
 // If user doesn't insert 4, then pass function will pass the string to the server  
 if(!message.equals("4")){  
 encryptedMessage = *sign*(message);  
 String pack = message + " " + encryptedMessage;  
 int result = *pass*(pack);  
 System.*out*.println("The result is " + result +".");  
 System.*out*.println();  
 }  
 }while(!message.equals("4"));  
 System.*out*.println("Client side quitting. The remote variable server is still running.");  
 } catch (IOException e) {  
 System.*out*.println("IO Exception:" + e.getMessage());  
 } finally {  
 try {  
 if (*clientSocket* != null) {  
 *clientSocket*.close();  
 }  
 } catch (IOException e) {  
 // ignore exception on close  
 }  
 }  
 }  
 // This function returns a string concatenating type, id, number  
 // And returns 4 if the user insert 4.  
 public static String getMessage(){  
 Scanner readInput = new Scanner(System.*in*);  
 String[] options = {"Add a value to your sum.","Subtract a value from your sum.","Get your sum.","Exit client."};  
 for(int i = 0; i < options.length; i++){  
 System.*out*.println((i+1) + ". " + options[i]);  
 }  
 int operator = Integer.*parseInt*(readInput.nextLine());  
 String number = "0";  
 // return string based on user's insertion.  
 switch(operator) {  
 // if the user insert 4, then return 4 to exit the client program  
 case 4:  
 return String.*valueOf*(operator);  
 case 1:  
 System.*out*.println("Enter value to add:");  
 number = readInput.nextLine();  
 break;  
 case 2:  
 System.*out*.println("Enter value to subtract:");  
 number = readInput.nextLine();  
 break;  
 default:  
 }  
 return *id* + "," + *e* + "," + *n* + "," + String.*valueOf*(operator) + "," + number;  
 }  
 //This function takes the concatenated string from the client and pass it to the server  
 public static int pass(String s){  
 String data = null;  
 try {  
 String signed\_s = *sign*(s);  
 *in* = new BufferedReader(new InputStreamReader(*clientSocket*.getInputStream()));  
 *out* = new PrintWriter(new BufferedWriter(new OutputStreamWriter(*clientSocket*.getOutputStream())));  
 *out*.println(s);  
 *out*.flush();  
 data = *in*.readLine(); // read a line of data from the stream  
 System.*out*.println("Received: " + data);  
 } catch (IOException e) {  
 System.*out*.println("IO Exception:" + e.getMessage());  
 }  
 return Integer.*parseInt*(data);  
 }  
 // Code Exploit from RSAExample,  
 // it will generate the public key and private key everytime the client starts the program  
 public static void generateKey(){  
 System.*out*.println("Generating keys");  
 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);  
 *public\_key* = String.*valueOf*(*e*) + String.*valueOf*(*n*);  
 *private\_key* = String.*valueOf*(*d*) + String.*valueOf*(*n*);  
 System.*out*.println("your public key = (" + *public\_key* + ")"); // Step 6: (e,n) is the RSA public key  
 System.*out*.println("your private key = (" + *private\_key* + ")"); // Step 7: (d,n) is the RSA private key  
 }  
 // This method generate a unique id with the last 20 byte of the public key  
 public static void generateID(){  
 try {  
 MessageDigest md = MessageDigest.*getInstance*("SHA-256");  
 md.update(*public\_key*.getBytes());  
 byte[] hash\_value = md.digest();  
 byte[] id\_byte = new byte[*NUMBER\_OF\_ID\_DIGITS*];  
 int len\_of\_hash\_value = hash\_value.length;  
 // copy the last 20 bytes to id\_byte  
 for(int i = 0; i < *NUMBER\_OF\_ID\_DIGITS*; i++){  
 id\_byte[*NUMBER\_OF\_ID\_DIGITS*-i-1] = hash\_value[len\_of\_hash\_value - i - 1];  
 }  
 *id* = new BigInteger(id\_byte);  
 System.*out*.println("Your id is: " + *id*);  
 }  
 catch(NoSuchAlgorithmException e) {  
 System.*out*.println("No Hash available" + e);  
 }  
 }  
 // compute the signature (the hash\_value of the whole message)  
 static public String sign(String message) {  
 // compute the digest with SHA-256  
 BigInteger c = null;  
 try{  
 byte[] bytesOfMessage = message.getBytes("UTF-8");  
 MessageDigest md = MessageDigest.*getInstance*("SHA-256");  
 byte[] bigDigest = md.digest(bytesOfMessage);  
 // we add a 0 byte as the most significant byte to keep  
 // the value to be signed non-negative.  
 // Copy every byte of bigDigest  
 byte[] messageDigest = new byte[bigDigest.length + 1];  
 messageDigest[0] = 0; // most significant set to 0  
 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  
 c = m.modPow(*d*, *n*);  
 // return this as a big integer string  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
 return c.toString();  
 }  
}

**Project2Task5Server**

import java.math.BigInteger;  
import java.net.\*;  
import java.io.\*;  
import java.security.MessageDigest;  
import java.util.Map;  
import java.util.Scanner;  
import java.util.TreeMap;  
// Name: Leo Lin  
// Andrew ID: hungfanl  
public class VerifyingServerTCP {  
 static BigInteger *n*, *e*, *id*;  
 static int *operator*;  
 static Map<BigInteger, Integer> *database* = new TreeMap<>();  
 public static void main(String args[]) {  
 Socket clientSocket = null;  
 try {  
 int serverPort = 7777; // the server port we are using  
 // Create a new server socket  
 ServerSocket listenSocket = new ServerSocket(serverPort);  
 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())));  
 // In order for the server to run forever, we need to handle the situation when client shut down  
 while (true) {  
 String data;  
 // When the client is connected, there will be string pass to in.  
 // Under the situation, we can safely conduct the calculation and return the value.  
 if(in.hasNextLine()){  
 data = in.nextLine();  
 // The string passed by the client is id, e, n, operator, number seperated by ","  
 // and the hash message seperated by " "  
 String[] message = data.split(" ");  
 String[] operation = message[0].split(",");  
 *id* = new BigInteger(operation[0]);  
 *e* = new BigInteger(operation[1]);  
 *n* = new BigInteger(operation[2]);  
 *operator* = Integer.*parseInt*(operation[3]);  
 int number = Integer.*parseInt*(operation[4]);  
 // conduct corresponding arithmetic according to the three index  
 // verify the hash message  
 if(*idMatch*(message[0], message[1])){  
 int outcome = *arithmetic*(*operator*, *id*, number);  
 // return the outcome of calculation to the client  
 System.*out*.println("Returning sum of " + outcome + " to client");  
 out.println(outcome);  
 }  
 else out.println("The request is wrongly encrypted");  
 out.flush();  
 }  
 // However, when the client is shut down, there will not be next line pass to in.  
 // In such cases, we will need to have clientSocket to accept another socket and renew the in/out  
 else {  
 clientSocket = listenSocket.accept();  
 in = new Scanner(clientSocket.getInputStream());  
 out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream())));  
 }  
 }  
 // Handle exceptions  
 } catch (IOException e) {  
 System.*out*.println("IO Exception:" + e.getMessage());  
  
 // If quitting (typically by you sending quit signal) clean up sockets  
 } finally {  
 try {  
 if (clientSocket != null) {  
 clientSocket.close();  
 }  
 } catch (IOException e) {  
 // ignore exception on close  
 }  
 }  
 }  
 // Doing corresponding arithmetic according to the user's request  
 public static int arithmetic(int operator, BigInteger id, int number){  
 String[] operation = {"Addition", "Subtraction", "Check"};  
 System.*out*.println("The visitor's ID is: " + id);  
 System.*out*.println("Operator: " + operation[operator-1]);  
 if(operator == 1){  
 *database*.put(id, *database*.getOrDefault(id,0) + number);  
 }  
 else if (operator == 2){  
 *database*.put(id, *database*.getOrDefault(id,0) - number);  
 }  
 // return the number after calculation  
 return *database*.getOrDefault(id,0);  
 }  
  
 // Check if the hash of messageToCheck is same as the decryption if encryptedHashStr  
 // Exploit from ShortMessageVerify.java provided by the handout  
 public static boolean idMatch(String messageToCheck, String encryptedHashStr) {  
 BigInteger decryptedHash = null;  
 BigInteger bigIntegerToCheck = null;  
 try{  
 // Decrypt it  
 decryptedHash = new BigInteger(encryptedHashStr).modPow(*e*, *n*);  
 // Get the bytes from messageToCheck  
 byte[] bytesOfMessageToCheck = messageToCheck.getBytes("UTF-8");  
 // compute the digest of the message with SHA-256  
 MessageDigest md = MessageDigest.*getInstance*("SHA-256");  
  
 byte[] bigDigest = md.digest(bytesOfMessageToCheck);  
  
 // messageToCheckDigest is a full SHA-256 digest  
 // add a zero byte in front of bigDigest  
 byte[] messageToCheckDigest = new byte[bigDigest.length + 1];  
 messageToCheckDigest [0] = 0; // most significant set to 0  
 for(int i = 0; i < bigDigest.length; i++){  
 messageToCheckDigest [i+1] = bigDigest[i];  
 }  
 bigIntegerToCheck = new BigInteger(messageToCheckDigest);  
 } catch (Exception e){  
 e.printStackTrace();  
 }  
 System.*out*.println("The hash message provided by the client: " + decryptedHash);  
 System.*out*.println("The hash value of the message:" + bigIntegerToCheck);  
 System.*out*.println("Verify result: " + bigIntegerToCheck.equals(decryptedHash));  
 return bigIntegerToCheck.equals(decryptedHash);  
 }  
}

**Project2Task5ClientConsole**

****

**Project2Task5ServerConsole**

