Distributed Systems
Project 2 Documentation
Sai Rajuladevi (srajulad)

Task 0:

Project2Task0ClientConsole

```
The client is running.
Input a server side port number: 6789
1
Reply: 1
2
Reply: 2
3
Reply: 3
4
Reply: 4
5
Reply: 5
halt!
Client side quitting

Process finished with exit code 0
```

Project2Task0ServerConsole

```
The server is running.
Input a server port number to listen on: 6789
Echoing: 1
Echoing: 2
Echoing: 3
Echoing: 4
Echoing: 5
Echoing: halt!
Server side quitting
Process finished with exit code 0
```

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

// modified from https://github.com/CMU-Heinz-95702/Project-2-Client-Server
public class EchoClientUDP{
    public static void main(String args[]){
        // args give message contents and server hostname
        System.out.println("The client is running.");
        DatagramSocket aSocket = null;
        try {
            // set to localhost to host on local machine and set port
            InetAddress aHost = InetAddress.getByName("localhost");
            int serverPort = 6789;

            // input server port to use
            System.out.print("Input a server side port number: ");
            Scanner readline = new Scanner(System.in);
            serverPort = readline.nextInt();

            // set up network socket to allow communication between server and client
            aSocket = new DatagramSocket();
            aSocket.setReuseAddress(true);
            String nextLine;// read console input
            BufferedReader typed = new BufferedReader(new InputStreamReader(System.in));
            while ((nextLine = typed.readLine()) != null) {
                byte [] m = nextLine.getBytes(); // convert console in into byte packets
                DatagramPacket request = new DatagramPacket(m,  m.length, aHost, serverPort);
                aSocket.send(request); // send
```

```
29 packet
30                 byte[] buffer = new byte[1000]; //
   create a buffer to receiver server packet
31                 DatagramPacket reply = new
   DatagramPacket(buffer, buffer.length); // format
   for receiving packet
32                 aSocket.receive(reply); // receive
   from socket
33                 String replyString = new String(
   reply.getData()).substring(0,reply.getLength());
   // get proper length of string
34
35                 // halt logic
36                 if(replyString.equalsIgnoreCase("
   halt!")){
37                     System.out.println("Client side
    quitting");
38                     break;
39                 }
40                 else{
41                     System.out.println("Reply: " +
   replyString); // send reply if not halt
42                 }
43             }
44
45             // catch potential exceptions
46         }catch (SocketException e) {System.out.
   println("Socket: " + e.getMessage());
47         }catch (IOException e){System.out.println("
   IO: " + e.getMessage());
48         }finally {if(aSocket != null) aSocket.close
   ();}
49     }
50 }
```

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

// modified from https://github.com/CMU-Heinz-95702/Project-2-Client-Server
public class EchoServerUDP{
    public static void main(String args[]){
        System.out.println("The server is running."); // lab instructions
        DatagramSocket aSocket = null;
        byte[] buffer = new byte[1000];  // set up packet buffer for client message
        try{
            // set up ports
            System.out.print("Input a server port number to listen on: ");
            Scanner readline = new Scanner(System.in);
            int serverPort = readline.nextInt(); // convert to int

            // set up sockets to receive client packets
            aSocket = new DatagramSocket(serverPort);
            aSocket.setReuseAddress(true);

            DatagramPacket request = new DatagramPacket(buffer, buffer.length); // syntax for buffer
            while(true){ // loop to continue until 'halt!' is sent
                aSocket.receive(request); // receive request, and format a reply
                DatagramPacket reply = new DatagramPacket(request.getData(),
                        request.getLength(), request.getAddress(), request.getPort()); // syntax for reply from request
                String requestString = new String(
```

```
26 request.getData()).substring(0,request.getLength
   ()); // proper length
27                 System.out.println("Echoing: "+
   requestString);
28                 aSocket.send(reply); // send back
   the reply
29
30                 // halt logic
31                 if(requestString.equalsIgnoreCase("
   halt!")){
32                     System.out.print("Server side
   quitting");
33                     break;
34                 }
35             }
36             // catch potential exceptions
37         }catch (SocketException e){System.out.
   println("Socket: " + e.getMessage());
38         }catch (IOException e) {System.out.println(
   "IO: " + e.getMessage());
39         }finally {if(aSocket != null) aSocket.close
   ();}
40     }
41 }
```

Task 1:

Project2Task1ThreeConsoles
Client using 6798:
Echo Server UDP

```
The server is running.
Input a server port number to listen on: 6789
Echoing: 1
Echoing: 2
Echoing: 3
Echoing: halt!
Server side quitting
Process finished with exit code 0
```

Eavesdropper

```
The eavesdropper is running.
Input a server port number to listen to: 6798
Input a server port number to masquerade as: 6789

From Client: 1
Sending to Server: 1
From Server: 1
Sending to client: 1

From Client: 2
Sending to Server: 2
From Server: 2
Sending to client: 2

From Client: 3
Sending to Server: 3
From Server: 3
Sending to client: 3

From Client: halt!
Sending to Server: halt!
From Server: halt!
Sending to client: halt!
************** halt! message arrived **************
```

Client

```
The client is running.
Input a server side port number: 6798
1
Reply: 1
2
Reply: 2
3
Reply: 3
halt!
Client side quitting
```

Client using 6789
Echo Server UDP

```
The server is running.
Input a server port number to listen on: 6789
Echoing: 1
Echoing: 2
Echoing: 3
Echoing: halt!
Server side quitting
Process finished with exit code 0
```

Eavesdropper

```
The eavesdropper is running.
Input a server port number to listen to: 6798
Input a server port number to masquerade as: 6789
```

Client

```
The client is running.
Input a server side port number: 6789
1
Reply: 1
2
Reply: 2
3
Reply: 3
halt!
Client side quitting

Process finished with exit code 0
```

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

public class EavesdropperUDP{
    public static void main(String args[]){
        System.out.println("The eavesdropper is running."); // lab instructions
        DatagramSocket listenSocket = null;
        DatagramSocket masqueradeSocket = null;

        byte[] buffer = new byte[1000]; // create a buffer to receiver server packet

        try{
            InetAddress aHost = InetAddress.getByName("localhost");

            // listen port code
            System.out.print("Input a server port number to listen to: ");
            Scanner readline = new Scanner(System.in);
            int listenPort = readline.nextInt(); // convert to int

            listenSocket = new DatagramSocket(listenPort);
            listenSocket.setReuseAddress(true);

            // masquerade port code
            System.out.print("Input a server port number to masquerade as: ");
            int masqueradePort = readline.nextInt(); // convert to int

            masqueradeSocket = new DatagramSocket();
            masqueradeSocket.setReuseAddress(true);

            DatagramPacket request = new
```

```
31 DatagramPacket(buffer, buffer.length); // syntax
   for buffer
32
33              while(true){ // loop to continue until
    'halt!' is sent
34                  // get response from  client
35                  listenSocket.receive(request);
36                  String requestString = new String(
   request.getData()).substring(0,request.getLength
   ());
37                  System.out.println("\nFrom Client
   : "+ requestString);
38
39                  // send to server
40                  byte [] m = requestString.getBytes
   (); // convert console in into byte packets
41                  DatagramPacket reply = new
   DatagramPacket(m, m.length, aHost, masqueradePort);
42                  masqueradeSocket.send(reply); //
   send packet
43
44                  System.out.println("Sending to
   Server: "+ requestString);
45
46                  // get response from server
47                  DatagramPacket response = new
   DatagramPacket(buffer, buffer.length); // syntax
   for buffer
48                  masqueradeSocket.receive(response);
49                  String responseString = new String(
   response.getData()).substring(0,response.getLength
   ());
50
51                  System.out.println("From Server: "
   + responseString);
52
53                  // send to client
54                  response = new DatagramPacket(
   response.getData(),  response.getLength(), request.
   getAddress(), request.getPort());
55                  listenSocket.send(response); //
```

```
55 send packet
56
57                 System.out.println("Sending to
   client: "+ responseString);
58
59                 // halt logic
60                 if(requestString.equalsIgnoreCase("
   halt!")){
61                     System.out.print(
   "**************");
62                     System.out.print(" halt!
   message arrived **************");
63                 }
64
65
66             }
67             // catch potential exceptions
68         }catch (SocketException e){System.out.
   println("Socket: " + e.getMessage());
69         }catch (IOException e) {System.out.println(
   "IO: " + e.getMessage());
70         }finally {
71             if(listenSocket != null) listenSocket.
   close();
72             if(masqueradeSocket != null)
   masqueradeSocket.close();
73         }
74     }
75 }
```

Task 2:

Project2Task2ClientConsole

```
The client is running.
Input a server side port number: 6789
1
The server returned: 1.
2
The server returned: 3.
-3
The server returned: 0.
4
The server returned: 4.
5
The server returned: 9.
halt!
Client side quitting.

Process finished with exit code 0
```

```
The client is running.
Input a server side port number: 6789
6
The server returned: 15.
7
The server returned: 22.
-8
The server returned: 14.
9
The server returned: 23.
10
The server returned: 33.
halt!
Client side quitting.

Process finished with exit code 0
```

Project2Task2ServerConsole

```
The server is running.
Input a server port number to listen on: 6789
Adding 1 to 0
Returning sum of 1 to client.

Adding 2 to 1
Returning sum of 3 to client.

Adding -3 to 3
Returning sum of 0 to client.

Adding 4 to 0
Returning sum of 4 to client.

Adding 5 to 4
Returning sum of 9 to client.

Adding 6 to 9
Returning sum of 15 to client.

Adding 7 to 15
Returning sum of 22 to client.

Adding -8 to 22
Returning sum of 14 to client.

Adding 9 to 14
Returning sum of 23 to client.

Adding 10 to 23
Returning sum of 33 to client.
```

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

public class AddingClientUDP{
    static InetAddress aHost;
    static int serverPort;
    static DatagramSocket aSocket;
    static String replyString = "";


    public static void main(String args[]){
        // args give message contents and server hostname
        System.out.println("The client is running.");
        try {
            // set to localhost to host on local machine and set port
            aHost = InetAddress.getByName("localhost");
            serverPort = 6789;

            // input server port to use
            System.out.print("Input a server side port number: ");
            Scanner readline = new Scanner(System.in);
            serverPort = readline.nextInt();

            String nextLine;
            BufferedReader typed = new BufferedReader(new InputStreamReader(System.in));
            while ((nextLine = typed.readLine()) != null) {
                // halt logic
                if(nextLine.equalsIgnoreCase("halt!")){
                    System.out.println("Client side quitting.");
                    break;
```

```
32                  }
33                  else{
34                      int result = add(Integer.
   parseInt(nextLine));
35                      System.out.println("The server
   returned: " + replyString + "."); // send reply if
   not halt
36                  }
37              }
38
39              // catch potential exceptions
40          }catch (SocketException e) {System.out.
   println("Socket: " + e.getMessage());
41          }catch (IOException e){System.out.println("
   IO: " + e.getMessage());
42          }finally {if(aSocket != null) aSocket.close
   ();}
43      }
44      public static int add(int i) throws IOException
    {
45          // set up network socket to allow
   communication between server and client
46          aSocket = new DatagramSocket();
47
48          byte [] m = String.valueOf(i).getBytes();
   // convert console in into byte packets
49
50          DatagramPacket request = new DatagramPacket
   (m,  m.length, aHost, serverPort);
51          aSocket.send(request); // send packet
52
53          byte[] buffer = new byte[1000]; // create a
    buffer to receiver server packet
54          DatagramPacket reply = new DatagramPacket(
   buffer, buffer.length); // format for receiving
   packet
55          aSocket.receive(reply); // receive from
   socket
56          replyString = new String(reply.getData()).
   substring(0,reply.getLength()); // get proper
   length of string
```

```
57
58          return Integer.parseInt(replyString);
59      }
60  }
```

```
 1 import java.net.*;
 2 import java.io.*;
 3 import java.util.Scanner;
 4
 5 public class AddingServerUDP{
 6     private static int sum = 0;
 7
 8     public static void main(String args[]){
 9         System.out.println("The server is running."
   ); // lab instructions
10         DatagramSocket aSocket = null;
11         byte[] buffer = new byte[1000];  // set up
   packet buffer for client message
12         try{
13             // set up ports
14             System.out.print("Input a server port
   number to listen on: ");
15             Scanner readline = new Scanner(System.
   in);
16             int serverPort = readline.nextInt();
   // convert to int
17
18             // set up sockets to receive client
   packets
19             aSocket = new DatagramSocket(serverPort
   );
20             aSocket.setReuseAddress(true);
21
22             DatagramPacket request = new
   DatagramPacket(buffer, buffer.length); // syntax
   for buffer
23             while(true){ // loop to continue until
    'halt!' is sent
24                 aSocket.receive(request); //
   receive request, and format a reply
25
26                 String requestString = new String(
   request.getData()).substring(0,request.getLength
   ()); // proper length
27
28                 System.out.println("Adding " +
```

```
28 requestString + " to " + String.valueOf(sum));
29
30                 sum += Integer.valueOf(
   requestString.toString());
31
32                 String replyString = String.valueOf
   (sum);
33
34                 byte [] m = replyString.getBytes
   (); // convert console in into byte packets
35                 DatagramPacket reply  = new
   DatagramPacket(m,  m.length, request.getAddress(),
   request.getPort());
36
37                 System.out.println("Returning sum
   of "+replyString + " to client.\n");
38                 aSocket.send(reply); // send back
   the reply
39
40                 // halt logic
41                 if(replyString.equalsIgnoreCase("
   halt!")){
42                     System.out.print("Server side
   quitting");
43                     break;
44                 }
45             }
46             // catch potential exceptions
47         }catch (SocketException e){System.out.
   println("Socket: " + e.getMessage());
48         }catch (IOException e) {System.out.println(
   "IO: " + e.getMessage());
49         }finally {if(aSocket != null) aSocket.close
   ();}
50     }
51 }
```

Task 3:

Project2Task3ClientConsole

```
The client is running.
Input a server side port number: 6789

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
1
Enter value to add:
10
Enter your ID:
1
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
Enter your ID:
1
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
Enter your ID:
1
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
1
Enter value to add:
10
Enter your ID:
2
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:
7
Enter your ID:
2
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
3
Enter your ID:
2
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:
16
Enter your ID:
3
The result is: 16

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
2
Enter value to subtract:
3
Enter your ID:
3
The result is: 13

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
3
The result is: 13
```

```
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
```

Run client again:

```
The client is running.
Input a server side port number: 6789

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
1
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
Enter your ID:
2
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
3
Enter your ID:
3
The result is: 13
```

Project2Task3ServerConsole

```
The server is running.
Input a server port number to listen on: 6789

VisitorID: 1
Operation #: 1
Return Variable 10

VisitorID: 1
Operation #: 2
Return Variable 5

VisitorID: 1
Operation #: 3
Return Variable 5

VisitorID: 2
Operation #: 1
Return Variable 10

VisitorID: 2
Operation #: 2
Return Variable 3

VisitorID: 2
Operation #: 3
Return Variable 3
```

```
VisitorID: 3
Operation #: 1
Return Variable 16

VisitorID: 3
Operation #: 2
Return Variable 13

VisitorID: 3
Operation #: 3
Return Variable 13

VisitorID: 1
Operation #: 3
Return Variable 5

VisitorID: 2
Operation #: 3
Return Variable 3

VisitorID: 3
Operation #: 3
Return Variable 13
```

```
 1 import java.net.*;
 2 import java.io.*;
 3 import java.util.Scanner;
 4
 5 public class RemoteVariableClientUDP{
 6     static InetAddress aHost;
 7     static int serverPort;
 8     static DatagramSocket aSocket;
 9     static String replyString = "";
10
11
12     public static void main(String args[]){
13         // args give message contents and server
   hostname
14         System.out.println("The client is running."
   );
15         try {
16             // set to localhost to host on local
   machine and set port
17             aHost = InetAddress.getByName("
   localhost");
18             serverPort = 6789;
19
20             // input server port to use
21             System.out.print("Input a server side
   port number: ");
22             Scanner readline = new Scanner(System.
   in);
23             serverPort = readline.nextInt();
24
25
26             String ID = "";
27             String operation = "";
28             String value = "";
29
30             String packet = "";
31
32             while (true) {
33                 System.out.println("\n1. Add a
   value to your sum.\n" +
34                         "2. Subtract a value from
```

```
34 your sum.\n" +
35                         "3. Get your sum.\n" +
36                         "4. Exit client");
37
38                 int in = readline.nextInt();
39
40                 if(in == 1){
41                     operation = "1 ";
42                     System.out.println("Enter value
    to add: ");
43                     value = String.valueOf(readline
   .nextInt());
44                     System.out.println("Enter your
   ID: ");
45                     ID = String.valueOf(readline.
   nextInt()) + " ";
46
47                     packet += operation + ID +
   value;
48
49                 }
50                 else if(in == 2){
51                     operation = "2 ";
52                     System.out.println("Enter value
    to subtract: ");
53                     value = String.valueOf(readline
   .nextInt());
54                     System.out.println("Enter your
   ID: ");
55                     ID = String.valueOf(readline.
   nextInt()) + " ";
56
57                     packet += operation + ID +
   value;
58
59                 }
60                 else if(in == 3){
61                     operation = "3 ";
62                     System.out.println("Enter your
   ID: ");
63                     ID = String.valueOf(readline.
```

```
63 nextInt()) + " ";
64                     value = "";
65
66                     packet += operation + ID +
   value;
67                 }
68                 else if(in == 4){
69                     System.out.println("Client
   side quitting. The remote variable server is still
    running."); // send reply if not halt
70                     break;
71                 }
72
73                 System.out.println("The result is
   : " + communicate(packet));
74
75                 packet = "";
76
77             }
78
79             // catch potential exceptions
80         }catch (SocketException e) {System.out.
   println("Socket: " + e.getMessage());
81         }catch (IOException e){System.out.println(
   "IO: " + e.getMessage());
82         }finally {if(aSocket != null) aSocket.
   close();}
83     }
84     public static String communicate(String in)
   throws IOException {
85
86         aSocket = new DatagramSocket();
87
88         // operation code
89         byte [] m = in.getBytes(); // convert
   console in into byte packets
90         DatagramPacket operationRequest = new
   DatagramPacket(m,  m.length, aHost, serverPort);
91         aSocket.send(operationRequest); // send
   packet
92
```

```
 93          byte[] buffer = new byte[1000]; // create
    a buffer to receiver server packet
 94          DatagramPacket reply = new DatagramPacket(
    buffer, buffer.length); // format for receiving
    packet
 95          aSocket.receive(reply); // receive from
    socket
 96          replyString = new String(reply.getData()).
    substring(0,reply.getLength()); // get proper
    length of string
 97
 98          return replyString;
 99      }
100 }
```

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

public class RemoteVariableServerUDP{
    private static int sum = 0;
    private static TreeMap<Integer, Integer> tree_map = new TreeMap<Integer, Integer>();

    static String instructions = "1. Add a value to your sum.\n" +
            "2. Subtract a value from your sum.\n" +
            "3. Get your sum.\n" +
            "4. Exit client";

    public static void main(String args[]){
        System.out.println("The server is running."); // lab instructions
        DatagramSocket aSocket = null;
        byte[] buffer = new byte[1000];  // set up packet buffer for client message
        try{
            // set up ports
            System.out.print("Input a server port number to listen on: ");
            Scanner readline = new Scanner(System.in);
            int serverPort = readline.nextInt(); // convert to int

            // set up sockets to receive client packets
            aSocket = new DatagramSocket(serverPort);
            aSocket.setReuseAddress(true);

            DatagramPacket request = new DatagramPacket(buffer, buffer.length); // syntax for buffer
```

```
30
31              int id;
32              int value;
33
34              while(true){ // loop to continue until
    'halt!' is sent
35
36                  // operation request and reply
37                  aSocket.receive(request); //
   receive request, and format a reply
38                  String requestString = new String(
   request.getData()).substring(0,request.getLength
   ()).strip(); // proper length
39
40                  String[] arrRequestString =
   requestString.split(" ");
41
42                  id = Integer.valueOf(
   arrRequestString[1]);
43
44                  if(!tree_map.containsKey(id)){
45                      if(arrRequestString[0].equals("
   1")){
46                          value = Integer.valueOf(
   arrRequestString[2]);
47                          tree_map.put(id, value);
48                      }
49                      else if(arrRequestString[0].
   equals("2")){
50                          value = -1*Integer.valueOf(
   arrRequestString[2]);
51                          tree_map.put(id, value);
52                      }
53                  }
54                  else{
55                      if(arrRequestString[0].equals("
   1")){
56                          value = Integer.valueOf(
   arrRequestString[2]);
57                          tree_map.replace(id,
   tree_map.get(id)+value);
```

```
58                      }
59                      else if(arrRequestString[0].
   equals("2")){
60                          value = Integer.valueOf(
   arrRequestString[2]);
61                          tree_map.replace(id,
   tree_map.get(id)-value);
62                      }
63                  }
64
65                  System.out.println("\nVisitorID: "
    + arrRequestString[1]);
66                  System.out.println("Operation #: "
    + arrRequestString[0]);
67                  System.out.println("Return Variable
    " + String.valueOf(tree_map.get(id)));
68
69
70                  byte [] m = (String.valueOf(
   tree_map.get(id))).getBytes(); // convert console
   in into byte packets
71
72                  DatagramPacket reply  = new
   DatagramPacket(m,  m.length, request.getAddress(),
   request.getPort());
73                  aSocket.send(reply); // send back
   the reply
74
75
76              }
77              // catch potential exceptions
78          }catch (SocketException e){System.out.
   println("Socket: " + e.getMessage());
79          }catch (IOException e) {System.out.println(
   "IO: " + e.getMessage());
80          }finally {if(aSocket != null) aSocket.close
   ();}
81      }
82 }
```

Task 4:

Project2Task4ClientConsole

```
The client is running.
Input a server side port number: 6789

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
1
Enter value to add:
10
Enter your ID:
1
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
Enter your ID:
1
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
Enter your ID:
1
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
1
Enter value to add:
45
Enter your ID:
2
The result is: 45

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:
75
Enter your ID:
2
The result is: -30

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
2
The result is: -30
```

```
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
1
Enter value to add:
63
Enter your ID:
3
The result is: 63

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:
3
The result is: 58

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
3
The result is: 58
```

```
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
4
Client side quitting. The remote variable server is still running.
```

Client rerun:

```
The client is running.
Input a server side port number: 6789

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
1
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
Enter your ID:
2
The result is: -30

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
Enter your ID:
3
The result is: 58
```

Project2Task4ServerConsole

```
The server is running.
Input a server port number to listen on: 6789

VisitorID: 1
Operation #: 1
Return Variable 10

VisitorID: 1
Operation #: 2
Return Variable 5

VisitorID: 1
Operation #: 3
Return Variable 5

VisitorID: 2
Operation #: 1
Return Variable 45

VisitorID: 2
Operation #: 2
Return Variable -30

VisitorID: 2
Operation #: 3
Return Variable -30
```

```
VisitorID: 3
Operation #: 1
Return Variable 63

VisitorID: 3
Operation #: 2
Return Variable 58

VisitorID: 3
Operation #: 3
Return Variable 58

VisitorID: 1
Operation #: 3
Return Variable 5

VisitorID: 2
Operation #: 3
Return Variable -30

VisitorID: 3
Operation #: 3
Return Variable 58
```

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

public class RemoteVariableClientTCP{
    static InetAddress aHost;
    static int serverPort;
    static Socket clientSocket = null;
    static String replyString = "";


    public static void main(String args[]){
        // args give message contents and server hostname
        System.out.println("The client is running." );
        try {
            // set to localhost to host on local machine and set port
            aHost = InetAddress.getByName("localhost");
            serverPort = 6789;

            // input server port to use
            System.out.print("Input a server side port number: ");
            Scanner readline = new Scanner(System.in);
            serverPort = readline.nextInt();

            clientSocket = new Socket("localhost", serverPort);


            String ID = "";
            String operation = "";
            String value = "";

            String packet = "";

            // create packet to server
```

```
35              while (true) {
36                  System.out.println("\n1. Add a
   value to your sum.\n" +
37                          "2. Subtract a value from
   your sum.\n" +
38                          "3. Get your sum.\n" +
39                          "4. Exit client");
40
41                  int in = readline.nextInt();
42
43                  if(in == 1){
44                      operation = "1 ";
45                      System.out.println("Enter value
    to add: ");
46                      value = String.valueOf(readline
   .nextInt());
47                      System.out.println("Enter your
   ID: ");
48                      ID = String.valueOf(readline.
   nextInt()) + " ";
49
50                      packet += operation + ID +
   value;
51
52                  }
53                  else if(in == 2){
54                      operation = "2 ";
55                      System.out.println("Enter value
    to subtract: ");
56                      value = String.valueOf(readline
   .nextInt());
57                      System.out.println("Enter your
   ID: ");
58                      ID = String.valueOf(readline.
   nextInt()) + " ";
59
60                      packet += operation + ID +
   value;
61
62                  }
63                  else if(in == 3){
```

```
64                      operation = "3 ";
65                      System.out.println("Enter your
    ID: ");
66                      ID = String.valueOf(readline.
   nextInt()) + " ";
67                      value = "";
68
69                      packet += operation + ID +
   value;
70                  }
71                  else if(in == 4){
72                      System.out.println("Client
   side quitting. The remote variable server is still
    running.");
73                      // send reply if not halt
74                      clientSocket.close();
75                      break;
76                  }
77
78                  System.out.println("The result is
   : " + communicate(packet));
79
80                  packet = "";
81
82
83              }
84
85              // catch potential exceptions
86          }catch (IOException e) {
87              System.out.println("IO Exception:" + e
   .getMessage());
88          } finally {
89              try {
90                  if (clientSocket != null) {
91                      clientSocket.close();
92                  }
93              } catch (IOException e) {
94                  // ignore exception on close
95              }
96          }
97
```

```
 98     }
 99     // ----------------------- Proxy style
    communication code -----------------------
100     // sends packet and reads reply
101     public static String communicate(String in)
    throws IOException {
102
103         BufferedReader read = new BufferedReader(
    new InputStreamReader(clientSocket.getInputStream
    ()));
104
105         PrintWriter out = new PrintWriter(new
    BufferedWriter(new OutputStreamWriter(clientSocket
    .getOutputStream())));
106
107         out.println(in);
108         out.flush();
109
110         String replyString = read.readLine();
111
112         return replyString;
113     }
114 }
```

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

public class RemoteVariableServerTCP{
    private static TreeMap<Integer, Integer> tree_map = new TreeMap<Integer, Integer>();

    static String instructions = "1. Add a value to your sum.\n" +
            "2. Subtract a value from your sum.\n" +
            "3. Get your sum.\n" +
            "4. Exit client";

    public static void main(String args[]){
        System.out.println("The server is running."); // lab instructions
        Socket clientSocket = null;
        byte[] buffer = new byte[1000];  // set up packet buffer for client message
        try{
            // set up ports
            System.out.print("Input a server port number to listen on: ");
            Scanner readline = new Scanner(System.in);
            int serverPort = readline.nextInt(); // convert to int

            ServerSocket listenSocket = new ServerSocket(serverPort);

            int id;
            int value;

            while(true){ // loop to continue until 'halt!' is sent
                clientSocket = listenSocket.accept();
```

```
31
32                  Scanner in;
33                  in = new Scanner(clientSocket.
   getInputStream());
34
35                  PrintWriter out;
36                  out = new PrintWriter(new
   BufferedWriter(new OutputStreamWriter(clientSocket.
   getOutputStream())));
37
38
39                  // operation request and reply
40                  while(in.hasNextLine()){
41                      String requestString = in.
   nextLine();
42
43                      String[] arrRequestString =
   requestString.split(" ");
44
45                      id = Integer.valueOf(
   arrRequestString[1]);
46
47                      // treemap logic to ensure
   unique id is generated
48                      if(!tree_map.containsKey(id)){
49                          if(arrRequestString[0].
   equals("1")){
50                              value = Integer.valueOf
   (arrRequestString[2]);
51                              tree_map.put(id, value
   );
52                          }
53                          else if(arrRequestString[0
   ].equals("2")){
54                              value = -1*Integer.
   valueOf(arrRequestString[2]);
55                              tree_map.put(id, value
   );
56                          }
57                          else{
58                              tree_map.put(id, 0);
```

```
58 // default for no id input
59                         }
60                     }
61                     else{
62                         if(arrRequestString[0].
   equals("1")){
63                             value = Integer.valueOf
   (arrRequestString[2]);
64                             tree_map.replace(id,
   tree_map.get(id)+value);
65                         }
66                         else if(arrRequestString[0
   ].equals("2")){
67                             value = Integer.valueOf
   (arrRequestString[2]);
68                             tree_map.replace(id,
   tree_map.get(id)-value);
69                         }
70                     }
71
72                     System.out.println("\nVisitorID
   : " + arrRequestString[1]);
73                     System.out.println("Operation
    #: " + arrRequestString[0]);
74                     System.out.println("Return
   Variable " + String.valueOf(tree_map.get(id)));
75
76
77                     out.println(String.valueOf(
   tree_map.get(id)));
78                     out.flush();
79                 }
80             }
81             // catch potential exceptions
82         } catch (IOException e) {
83             System.out.println("IO Exception:" + e.
   getMessage());
84
85             // If quitting (typically by you
   sending quit signal) clean up sockets
86         } finally {
```

```
87                 try {
88                     if (clientSocket != null) {
89                         clientSocket.close();
90                     }
91                 } catch (IOException e) {
92                     // ignore exception on close
93                 }
94             }
95         }
96 }
```

Task 5:

Project2Task5ClientConsole 1
Client 1

```
The client is running.

RSA public key: (65537,34379429750021483563
RSA private key: (21683533404292812046500345

Input a server side port number: 6789

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

Project2Task5ServerConsole 1
Server Out 1

```
The server is running.
Input a server port number to listen on: 6789

Visitor Public Key: (65537,343794297500214835633601830867603678
VisitorID: -284670005484759113400078042694246176017587700889
ID and Signature Verified: Yes
Operation #: 1
Return Variable 10

Visitor Public Key: (65537,343794297500214835633601830867603678
VisitorID: -284670005484759113400078042694246176017587700889
ID and Signature Verified: Yes
Operation #: 2
Return Variable 5

Visitor Public Key: (65537,343794297500214835633601830867603678
VisitorID: -284670005484759113400078042694246176017587700889
ID and Signature Verified: Yes
Operation #: 3
Return Variable 5
```

Project2Task5ClientConsole 2
Client 2

```
The client is running.

RSA public key: (65537,2817043330487931
RSA private key: (133620106156137496670

Input a server side port number: 6789

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
1
Enter value to add:
46
The result is: 46

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:
42
The result is: 4

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
3
The result is: 4
```

```
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
4
Client side quitting. The remote variable server is still running.

Process finished with exit code 0
```

Project2Task5ServerConsole 2
Server Out 2

```
Visitor Public Key: (65537,2817043330487931261443148904
VisitorID: -183372988221911332593995256656271889998219
ID and Signature Verified: Yes
Operation #: 1
Return Variable 46

Visitor Public Key: (65537,2817043330487931261443148904
VisitorID: -183372988221911332593995256656271889998219
ID and Signature Verified: Yes
Operation #: 2
Return Variable 4

Visitor Public Key: (65537,2817043330487931261443148904
VisitorID: -183372988221911332593995256656271889998219
ID and Signature Verified: Yes
Operation #: 3
Return Variable 4
```

Project2Task5ClientConsole 3
Client 3

```
The client is running.

RSA public key: (65537,400451032599682380971562
RSA private key: (268236261655148341276998157 91

Input a server side port number: 6789

1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
1
Enter value to add:
234
The result is: 234

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:
7654
The result is: -7420

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: -7420
```

```
1. Add a value to your sum.
2. Subtract a value from your sum.
3. Get your sum.
4. Exit client
4
Client side quitting. The remote variable server is still running.

Process finished with exit code 0
```

Project2Task5ServerConsole 3
Server Out 3

```
Visitor Public Key: (65537,4004510325996
VisitorID: -568344306407253318584412588 5
ID and Signature Verified: Yes
Operation #: 1
Return Variable 234

Visitor Public Key: (65537,4004510325996
VisitorID: -568344306407253318584412588 5
ID and Signature Verified: Yes
Operation #: 2
Return Variable -7420

Visitor Public Key: (65537,4004510325996
VisitorID: -568344306407253318584412588 5
ID and Signature Verified: Yes
Operation #: 3
Return Variable -7420
```

```java
import java.math.BigInteger;
import java.net.*;
import java.io.*;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;

public class SigningClientTCP{
    static InetAddress aHost;
    static int serverPort;
    static Socket clientSocket = null;
    static String replyString = "";


    public static void main(String args[]) throws NoSuchAlgorithmException {
        System.out.println("The client is running.\n");

        // ---------------------- Generate the key ----------------------
        // from https://github.com/CMU-Heinz-95702/Project-2-Client-Server

        // 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.
```

```
32          // Change 400 to 2048, recompile, and run
   the program again and you will
33          // notice it takes much longer to do the
   math with that many bits.
34          BigInteger p = new BigInteger(400, 100, rnd
   );
35          BigInteger q = new BigInteger(400, 100, rnd
   );
36
37          // Step 2: Compute n by the equation n = p
    * q.
38          n = p.multiply(q);
39
40          // Step 3: Compute phi(n) = (p-1) * (q-1)
41          BigInteger phi = (p.subtract(BigInteger.ONE
   )).multiply(q.subtract(BigInteger.ONE));
42
43          // Step 4: Select a small odd integer e
   that is relatively prime to phi(n).
44          // By convention the prime 65537 is used as
    the public exponent.
45          e = new BigInteger("65537");
46
47          // Step 5: Compute d as the multiplicative
   inverse of e modulo phi(n).
48          d = e.modInverse(phi);
49
50          System.out.println("RSA public key: (" + e
    + "," + n + ")");  // Step 6: (e,n) is the RSA
   public key
51          System.out.println("RSA private key: (" + d
    + "," + n + ")\n");  // Step 7: (d,n) is the RSA
   private key
52
53
54          // ---------------------- Generate the ID
    ----------------------
55          // from https://github.com/CMU-Heinz-95702/
   Project-2-Client-Server
56
57          String rawPublicKey = String.valueOf(e) +
```

```
57 String.valueOf(n);
58
59          MessageDigest md  = MessageDigest.
   getInstance("SHA-256");
60          md.update(rawPublicKey.getBytes());
61
62          byte[] digest = md.digest();
63
64          BigInteger biID = new BigInteger(Arrays.
   copyOfRange(digest, digest.length-20, digest.length
   ));
65          String ID = biID.toString() + " ";
66          // source: https://stackoverflow.com/
   questions/18367539/slicing-byte-arrays-in-java
67
68
69          // ---------------------- Connect socket
   and create packet to send ----------------------
70          try {
71              // set to localhost to host on local
   machine and set port
72              aHost = InetAddress.getByName("
   localhost");
73              serverPort = 6789;
74
75              // input server port to use
76              System.out.print("Input a server side
   port number: ");
77              Scanner readline = new Scanner(System.
   in);
78              serverPort = readline.nextInt();
79
80              clientSocket = new Socket("localhost",
   serverPort);
81
82
83              String operation = "";
84              String value = "";
85
86              String packet = "";
87
```

```
 88              while (true) {
 89                  System.out.println("\n1. Add a
    value to your sum.\n" +
 90                          "2. Subtract a value from
    your sum.\n" +
 91                          "3. Get your sum.\n" +
 92                          "4. Exit client");
 93
 94                  int in = readline.nextInt();
 95
 96                  if(in == 1){
 97                      operation = "1 ";
 98                      System.out.println("Enter
    value to add: ");
 99                      value = String.valueOf(
    readline.nextInt()) + " ";
100                  }
101                  else if(in == 2){
102                      operation = "2 ";
103                      System.out.println("Enter
    value to subtract: ");
104                      value = String.valueOf(
    readline.nextInt() + " ");
105                  }
106                  else if(in == 3){
107                      operation = "3 ";
108                      value = "| ";
109                  }
110                  else if(in == 4){
111                      System.out.println("Client
    side quitting. The remote variable server is still
     running."); // send reply if not halt
112                      break;
113                  }
114
115                  packet += operation + ID + value
     + e + " " + n;
116
117                  // compute the digest with SHA-256
118                  byte[] bytesOfMessage = packet.
    getBytes("UTF-8");
```

```
119                 MessageDigest md2 = MessageDigest.
    getInstance("SHA-256");
120                 byte[] bigDigest = md2.digest(
    bytesOfMessage);
121                 byte[] messageDigest = new byte[
    bigDigest.length+1];
122
123                 // we only want two bytes of the
    hash for ShortMessageSign
124                 // we add a 0 byte as the most
    significant byte to keep
125                 // the value to be signed non-
    negative.
126                 messageDigest[0] = 0;   // most
    significant set to 0
127                 for(int i = 0; i < bigDigest.
    length; i++){
128                     messageDigest[i+1] = bigDigest
    [i];
129                 }
130
131                 // From the digest, create a
    BigInteger
132                 BigInteger m = new BigInteger(
    messageDigest);
133
134                 // encrypt the digest with the
    private key
135                 BigInteger c = m.modPow(d, n);
136
137                 // send the packet
138                 packet += " " + c.toString();
139                 System.out.println("The result is
    : " + communicate(packet));
140
141                 packet = "";
142             }
143
144             // catch potential exceptions
145         }catch (IOException ex) {
146             System.out.println("IO Exception:" +
```

```
146 ex.getMessage());
147         } finally {
148             try {
149                 if (clientSocket != null) {
150                     clientSocket.close();
151                 }
152             } catch (IOException ex) {
153                 // ignore exception on close
154             }
155         }
156     }
157 
158     // ---------------------- Proxy style
    communication code ----------------------
159     public static String communicate(String in)
    throws IOException {
160         BufferedReader read = new BufferedReader(
    new InputStreamReader(clientSocket.getInputStream
    ()));
161 
162         PrintWriter out = new PrintWriter(new
    BufferedWriter(new OutputStreamWriter(clientSocket
    .getOutputStream())));
163 
164         out.println(in);
165         out.flush();
166 
167         String replyString = read.readLine();
168 
169         return replyString;
170     }
171 }
```

```
 1 import java.math.BigInteger;
 2 import java.net.*;
 3 import java.io.*;
 4 import java.security.MessageDigest;
 5 import java.security.NoSuchAlgorithmException;
 6 import java.util.Scanner;
 7 import java.util.*;
 8
 9 public class VerifyingServerTCP{
10     private static TreeMap<String, Integer>
   tree_map = new TreeMap<>();
11
12     static String instructions = "1. Add a value to
    your sum.\n" +
13             "2. Subtract a value from your sum.\n"
    +
14             "3. Get your sum.\n" +
15             "4. Exit client";
16
17     public static void main(String args[]){
18         System.out.println("The server is running."
   ); // lab instructions
19
20
21         Socket clientSocket = null;
22         byte[] buffer = new byte[1000];  // set up
   packet buffer for client message
23         try{
24             // set up ports
25             System.out.print("Input a server port
   number to listen on: ");
26             Scanner readline = new Scanner(System.
   in);
27             int serverPort = readline.nextInt();
   // convert to int
28
29             ServerSocket listenSocket = new
   ServerSocket(serverPort);
30
31
32             String id;
```

```
33              int value;
34              String e;
35              String n;
36
37              while(true){ // loop to continue until
    'halt!' is sent
38                  clientSocket = listenSocket.accept
   ();
39
40                  Scanner in;
41                  in = new Scanner(clientSocket.
   getInputStream());
42
43                  PrintWriter out;
44                  out = new PrintWriter(new
   BufferedWriter(new OutputStreamWriter(clientSocket.
   getOutputStream())));
45
46                  // operation request and reply
47                  while(in.hasNext()){
48                      String requestString = in.
   nextLine();
49
50                      String[] arrRequestString =
   requestString.split(" ");
51
52                      // check id verification
53                      boolean idSignatureCheck =
   idSignatureCheck(arrRequestString[0],
   arrRequestString[1], arrRequestString[2],
54                              arrRequestString[3],
   arrRequestString[4], arrRequestString[5]);
55
56                      if(!idSignatureCheck){
57                          out.println("Error in
   request.");
58                          out.flush();
59                      }
60
61                      else{
62                          id = arrRequestString[1];
```

```
63
64                          if(!tree_map.containsKey(
   id)){
65                              if(arrRequestString[0
   ].equals("1")){
66                                  value = Integer.
   valueOf(arrRequestString[2]);
67                                  tree_map.put(id,
   value);
68                              }
69                              else if(
   arrRequestString[0].equals("2")){
70                                  value = -1*Integer
   .valueOf(arrRequestString[2]);
71                                  tree_map.put(id,
   value);
72                              }
73                              else{
74                                  tree_map.put(id, 0
   );
75                              }
76                          }
77                          else{
78                              if(arrRequestString[0
   ].equals("1")){
79                                  value = Integer.
   valueOf(arrRequestString[2]);
80                                  tree_map.replace(
   id, tree_map.get(id)+value);
81                              }
82                              else if(
   arrRequestString[0].equals("2")){
83                                  value = Integer.
   valueOf(arrRequestString[2]);
84                                  tree_map.replace(
   id, tree_map.get(id)-value);
85                              }
86                          }
87
88                          System.out.println("\n
   Visitor Public Key: (" + arrRequestString[3] + ","
```

```
 88  + arrRequestString[4] + ")");
 89                          System.out.println("
    VisitorID: " + arrRequestString[1]);
 90                          System.out.println("ID and
     Signature Verified: Yes");
 91                          System.out.println("
    Operation #: " + arrRequestString[0]);
 92                          System.out.println("Return
     Variable " + String.valueOf(tree_map.get(id)));
 93
 94                          out.println(String.valueOf
    (tree_map.get(id)));
 95                          out.flush();
 96                      }
 97                  }
 98              }
 99              // catch potential exceptions
100          } catch (IOException e) {
101              System.out.println("IO Exception:" + e
    .getMessage());
102              // If quitting (typically by you
    sending quit signal) clean up sockets
103          } catch (NoSuchAlgorithmException e) {
104              throw new RuntimeException(e);
105          } finally {
106              try {
107                  if (clientSocket != null) {
108                      clientSocket.close();
109                  }
110              } catch (IOException e) {
111                  // ignore exception on close
112              }
113          }
114      }
115
116 //    operation + ID + value + e + " " + n;
117      public static boolean idSignatureCheck(String
    operation, String id, String value, String e,
    String n, String signature) throws IOException,
    NoSuchAlgorithmException {
118
```

```
119         // ----------------- Check ID
     -----------------
120         String rawPublicKey = e + n;
121
122         MessageDigest md  = MessageDigest.
    getInstance("SHA-256");
123         md.update(rawPublicKey.getBytes());
124
125         byte[] digest = md.digest();
126
127         BigInteger idCheckBigInt = new BigInteger(
    Arrays.copyOfRange(digest, digest.length-20,
    digest.length));
128         String idCheck = idCheckBigInt.toString();
129
130
131         // ----------------- Check Message
     -----------------
132
133         // Take the encrypted string and make it a
     big integer
134         BigInteger encryptedHash = new BigInteger(
    signature);
135         BigInteger decryptedHash = encryptedHash.
    modPow(new BigInteger(e), new BigInteger(n));
136
137         // Get the bytes from messageToCheck
138         String messageToCheck = operation + " " +
    id + " " + value + " " + e + " " + n;
139
140         MessageDigest md2 = MessageDigest.
    getInstance("SHA-256");
141
142         byte[] messageToCheckDigest = md2.digest(
    messageToCheck.getBytes("UTF-8"));
143
144         // messageToCheckDigest is a full SHA-256
    digest
145         // take two bytes from SHA-256 and add a
    zero byte
146         byte[] extraByte = new byte[
```

```
146 messageToCheckDigest.length+1];
147
148         extraByte[0] = 0;   // most significant
    set to 0
149         for(int i = 1; i < extraByte.length; i++){
150             extraByte[i] = messageToCheckDigest[i-
    1];
151         }
152
153         // Make it a big int
154         BigInteger bigIntegerToCheck = new
    BigInteger(extraByte);
155
156         // inform the client on how the two
    compare
157         if((bigIntegerToCheck.compareTo(
    decryptedHash) == 0) && (idCheck.equals(id))) {
158             return true;
159         }
160         else{
161             return false;
162         }
163     }
164 }
```