A .JAMES CLARK SCHOOL OF ENGINEERING

MS TELECOMMUNICATION

NETWORKS AND PROTOCOLS -I

PROJECT REPORT

NIKITA NAIK

UID:114934819

PAVITHRA EZHILARASAN

UID:114842763

Problem Definition:

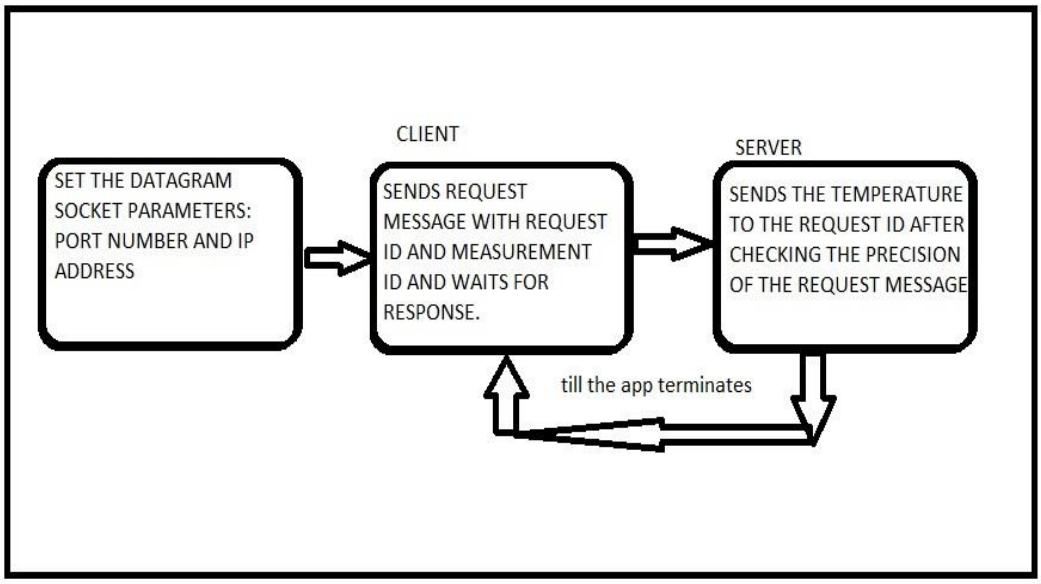
To establish a reliable communication link between the Java applications-client and the server, using the unreliable transport layer protocol, UDP. Since the services offered by UDP are unreliable, we have to use integrity checks at either ends of the link.The communication is an exchange of the temperature value corresponding to the measurement id entered by the user using the data.txt file.The file contains 100 lines, and each line represents a separate measurement ID/measurement value pair. The first number in each line is the measurement ID, which is a 16-bit unsigned integer. The second number is the corresponding temperature measurement value, which is a fixed-point decimal number with two fractional digits.At the client side, the machine will send a measurement id to the server machine for the corresponding temperature value. The server should use the content of the data.text file and provide the temperature value to the client in degrees Fahrenheit.

Design and Solution

To facilitate communication between the end-points of the communication link, the underlying transport layer protocol used is the User Datagram Protocol.Most of the real world applications use this protocol because of its low latency and low overhead properties over TCP.UDP provides port numbers to help distinguish different users.

Block diagram;

The overall flow of our analysis is shown in the block diagram below:



Setting the Datagram Socket parameters:

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.The server continuously receives datagram packets over a datagram socket. Each datagram packet received by the server indicates a client request for a quotation. When the server receives a datagram, it replies by sending a datagram packet that contains a one-line "quote of the moment" back to the client.

The general format:

//DatagramSocket(int port, InetAddress address);

//DatagramPacket(byte[] barr, int length, InetAddress address, int port);

So initially to begin with, we set the socket parameters-fixed port number and the ip address .The ip address is obtained using the getLocalHost().

The data.txt file is stored in a directory in the server machine for access to the data.

Operations performed on the client side:

In the Main method:

Sending request to the server.

- The client requests the server for the temperature value corresponding to the measurement id it sent.To ensure that we receive the desired value, we have designed a request message block.Everytime the client requests for a temperature value, it will do so by arranging the request in this block representation as a string.The block includes the request id and the measurement id along with the checksum value for the message.
- <request>

  <id>request id</id>

  <measurement>measurement id</measurement>

  </request>

  checksum

- The request id is used to ensure that the server temperature values for the measurement id in that particular request.Here the request id is generated randomly using the Math.random() .The measurement id is selected from the available pool of ids in the data.txt file.The checksum is calculated by calling the integritycheckfunction.The checksum is calculated over all

characters in the request message (except for the checksum), starting with the "<" character of the opening <request> tag, and ending with the ">" character of the closing </request> tag.

- The string message is then converted to a byte array using the getBytes method and stored inside byte[]b1. This byte array is then send to the server.After sending this byte array using the datagramsocket send function, we start a timer with an initial timeout value of 1 second.The timer is set using the datagramsocket.setSoTimeout().

Receiving response from the server

- The operation of the protocol used in this program is similar to the stop and wait protocol of sliding windows algorithm.After sending the data, the client waits for response from the server.The time it waits for the response is stated in the datagramsocket.setSoTimeout().In this case, it is 1 second that is 1000 milliseconds.
- So we receive the data here using the try and catch blocks.
- We start a while loop such that while the server still sends the data, the client receives the byte array sent by the server in the try block using the datagramsocket.receive method in a byte array and converts it back to a string and comes out of the loop.
- However, if the response is not received and the timer expires, the cursor moves into the catch block and resends the request data and increments the timer by a factor of 2 and also increments the counter.The counter is initialized to keep a count of the number of timeout sessions.
- Now when the counter exceeds a count of 3, the system exits out of the while loop giving out an Error connection failure message.
- Also in cases where the data is received not in the first timeout but in the successive sessions, then in that case after the data is received , the timer and the counter values are reset.
- Once the response message is received,we calculate the checksum using the integritycheckfunction and match it to the checksum value extracted from the response message.
- If the values don't match the client sends the request message again.
- After the integrity check, we extract the errorcodes from the response message that indicate the presence or absence of errors and accordingly print a message to the screen.
- When the received response is corrupted,we ask the user to resend the request if they wish to.If the user enters a positive response,we loop back to selecting a measurement id.
- Until the application is terminated,the entire code runs.

Operations performed on the server side

In the Main Method

RECEIVING REQUEST FROM THE CLIENT

While the client is sending the request , the server receives it using the datagramsocketreceive function and stores it in a byte array.It then converts it to String format.

- The request message is further checked to see if it was transmitted without errors by calculating the checksum by calling the integritycheckfunction.The checksum is calculated for the entire string except for the checksum part.
- The calculated checksum value is then compared against the received value, if the values dont match, the server throws an error an error code stating the received request is corrupted.Here the error code is code1.But if the values match, we check for the syntax of the message.For this we store a reference string at the server side.
- In this part if the syntax is not correct, the server throws an error code as code2.
- But if the syntax is the same as expected, we check for the measurement id value.Whether it is from the data.txt file.If not , the server throws an error code of code3.
- To check these values , we first extract the measurement id from the received request message.If the value is valid , the server throws an error code of code0 indicating that the request message was received without any errors.
- Now after all the functions performed, we extract the request id and the measurement id to generate the correct temperature value .
- All the error codes are arranged in an if else loop such that only if it passes the code1,it will be transferred for the second code and then the the code3.If it passes all three codes ,it is declared to be a error free message.

Sending the response message to the client

- Like on the client side the server also sends the response message in a a particular block format.The block includes the request id for the communication ,the measurement id from the client,the corresponding temperature value .
- For code0:
- <response>

  <id>request id</id>

<code>error code</code>

<measurement>measurement id</measurement>

<value>temperature value</value>

</response>

Checksum.

- We generate the response string and then convert it into a byte array.
- The above message is generated if the response code is code0
- However , if that is not the case,the response message includes only the request id ,the error code and the checksum field.
- For all other codes:
- <response>

<id>request id</id>

<code>error code</code>

</response>

Checksum

The various Error codes that need to be calculated and their explanation is as follows

RESPONSE CODE	MEANING
CODE 0	OK. The response has been created according to the request.
CODE 1	Error: integrity check failure. The request has one or more bit errors.
CODE 2	Error: malformed request. The syntax of the request message is not correct.
CODE 3	Error: non-existent measurement. The measurement with the requested measurement ID does not exist.

Integrity checksum function:

The integrity checksum is calculated on a 16-bit word sequence. The characters in a string are first converted to the binary equivalent of their ASCII values. The binary values of the first two characters are combined to form a 16 bit word, with the first character forming the Most significant byte and the second character forming the Least significant byte of the 16-bit word. This process is repeated for all the characters in a string. If the string contains odd number of characters, the Least significant byte of the last 16-bit word is taken as 0. Now, the 16-bit words are run in a for-loop with constants C=7919, D=65536. An XOR operation is done on each element of the 16-bit word with a variable S and this is further multiplied with constant C and modulus of D operation is done on it. At the end of the loop, the checksum value is stored in variable S. Now, this value is to be checked against the checksum value already on the message received/transmitted. If equal, the response code value is assigned as 0, else it is assigned as 1.

Syntax check:

The received string from the client side is to be checked for syntax at the server side. The string is taken without the checksum value. A reference string is stored which is compared against the string that is received. For every string comparison that is done right, a counter is updated. At the end of the operation, the counter is checked if it is equal to the total

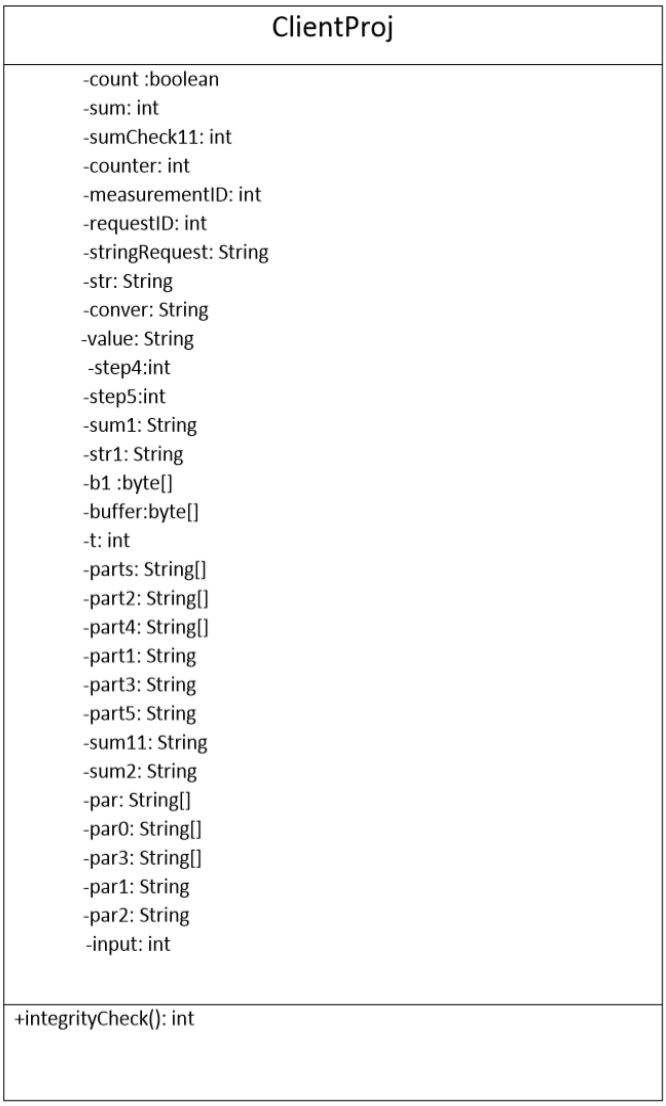
number of strings that should have been compared. If so, the response code is assigned as 0, else it is assigned as 2.

Checking for the measurement id :

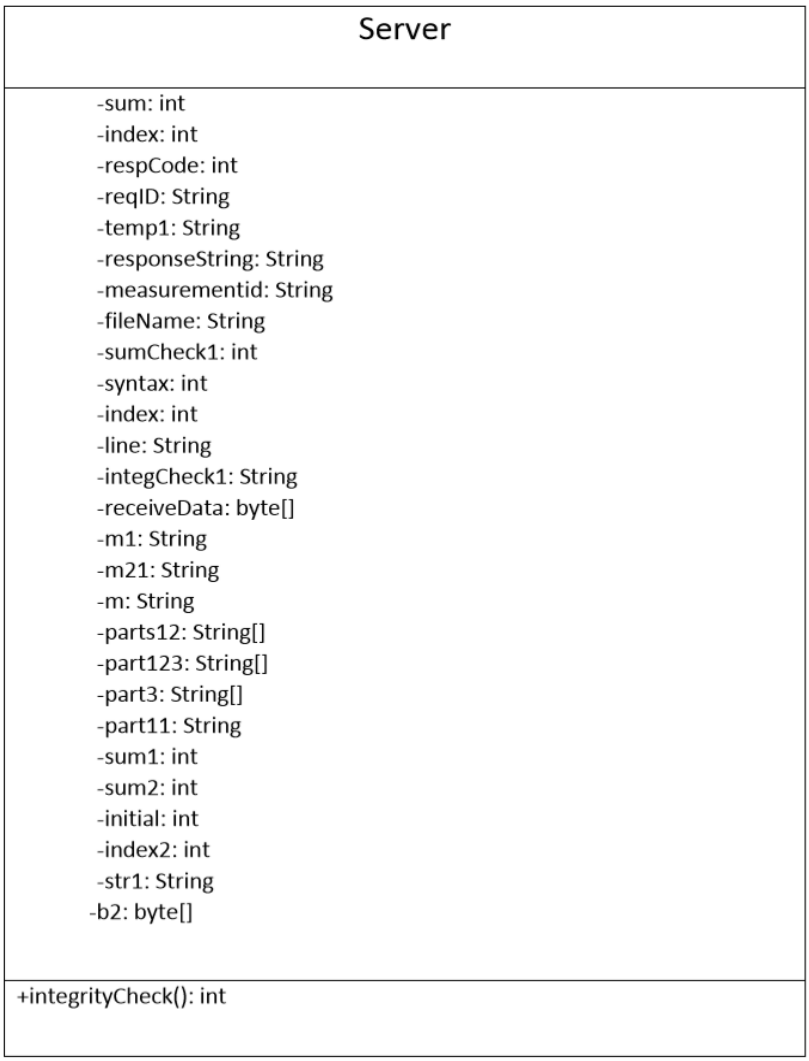
To open and read data from text file we used the filereader and buffered reader classes.Using a while loop such that it reads through the entire file and stores the read data ,each line in an array.Then after extracting the measurement id from the request message ,it will match it with every value in the file till the time it finds the exact match .If not,it displays code 3.

UML Diagram

At the client:



At the server:



STATE TRANSITION DIAGRAM

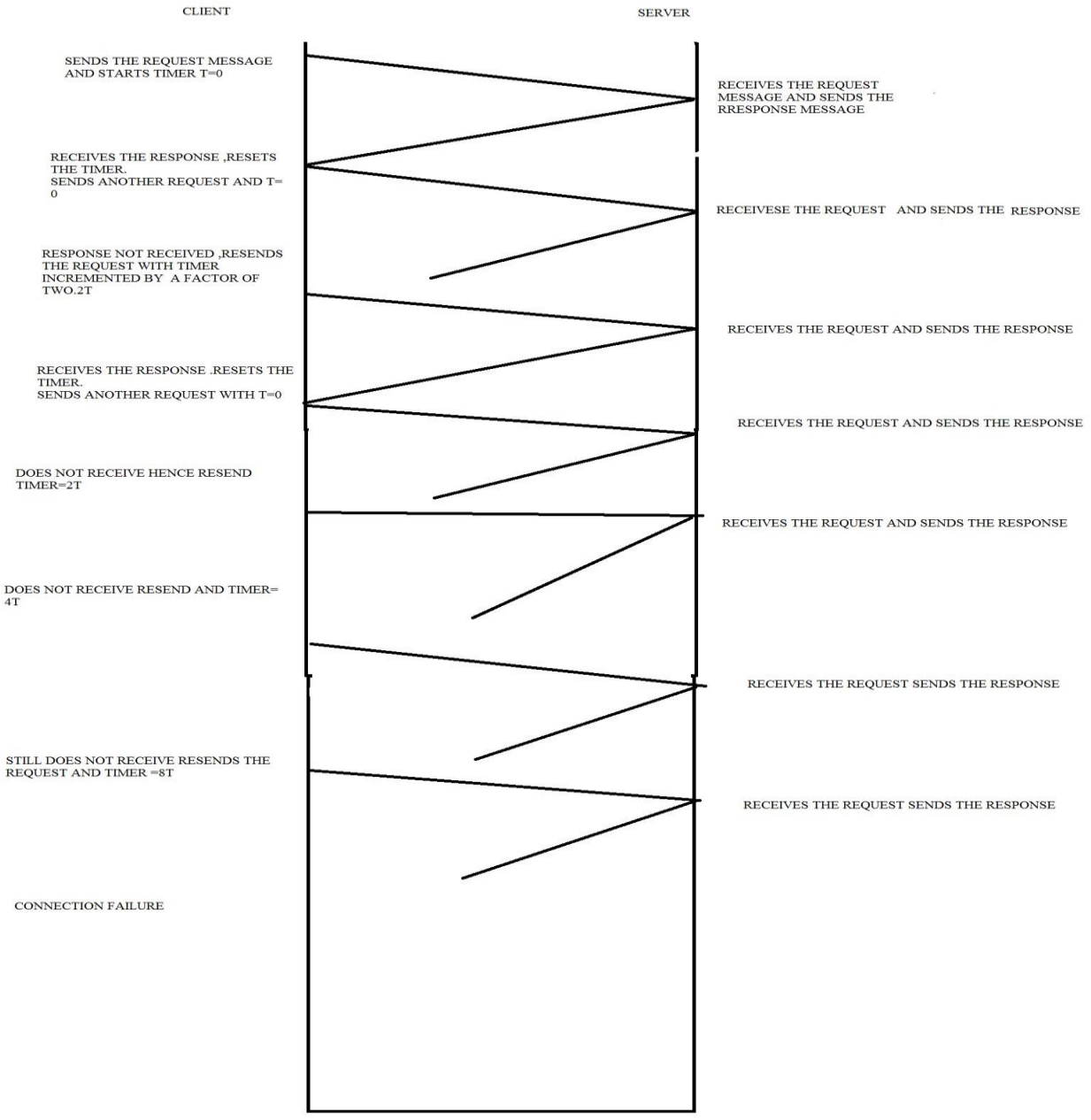
A State transition diagram represents the way the client and server applications behave during the communication process.

In this case, the protocol is similar to the stop and wait protocol of tcp ip.

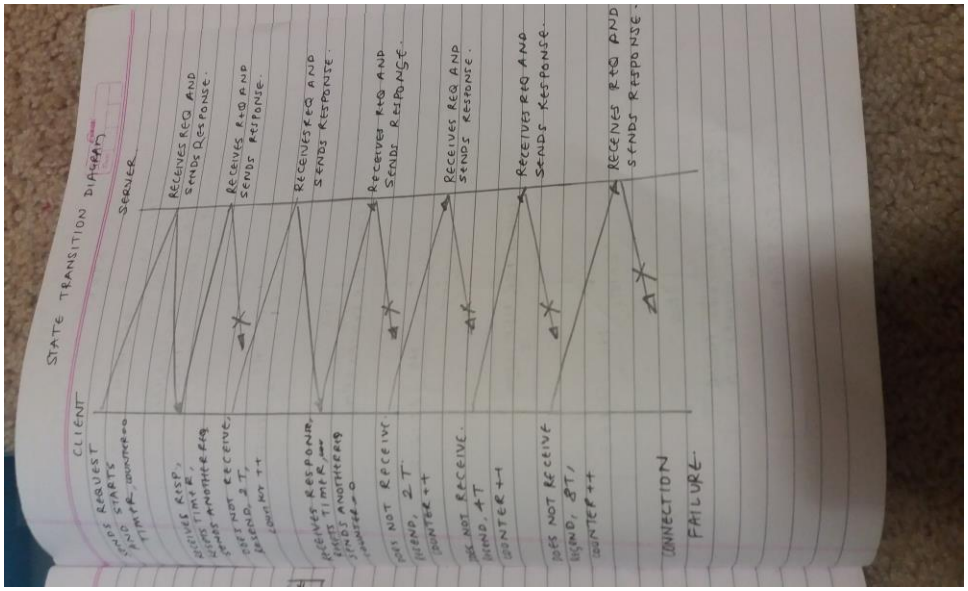
However, the underlying transport layer protocol used by us is the User Datagram protocol.

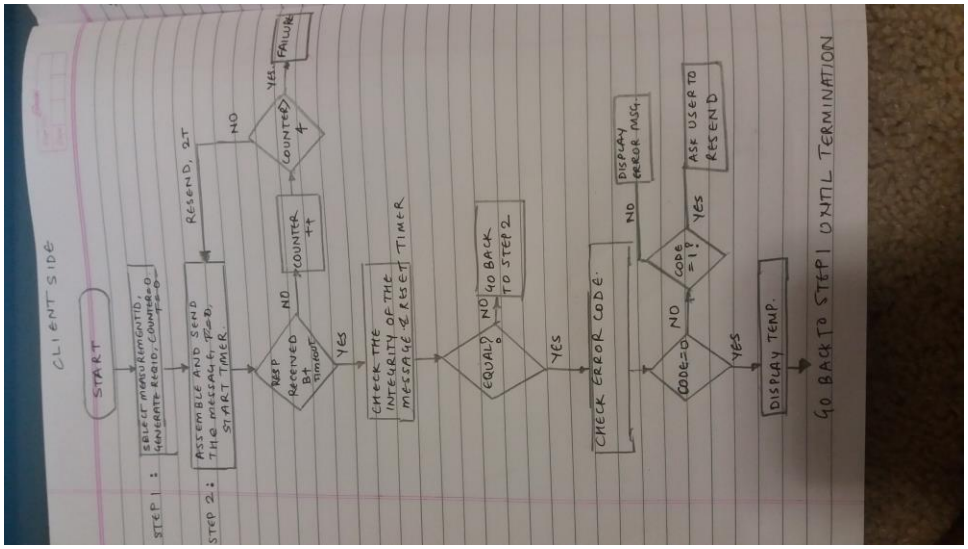
Defining the protocol operation

The client sends the request message to the server and waits for a response.the time duration for which it waits is 1000milliseconds.That is it starts a timer.If the client does not receive the message before the timer expires, it retransmits the message and increments the timer by a factor of 2 everytime.However,this process of retransmission stops after the 4^{th} timeout event and the client gives an error message indicating connection failure.If however the message is received not in the first but successive timer duration, then it resets the timer and the counter for the number of timeout events.


CLIENT
SERVER
SENDS THE REQUEST MESSAGE AND STARTS TIMER T=0
RECEIVES THE REQUEST MESSAGE AND SENDS THE RRESPONSE MESSAGE
RECEIVES THE RESPONSE ,RESETS THE TIMER.
SENDS ANOTHER REQUEST AND T= 0
RECEIVESE THE REQUEST AND SENDS THE RESPONSE
RESPONSE NOT RECEIVED ,RESENDS THE REQUEST WITH TIMER INCREMENTED BY A FACTOR OF TWO.2T
RECEIVES THE REQUEST AND SENDS THE RESPONSE
RECEIVES THE RESPONSE .RESETS THE TIMER.
SENDS ANOTHER REQUEST WITH T=0
RECEIVES THE REQUEST AND SENDS THE RESPONSE
DOES NOT RECEIVE HENCE RESEND TIMER=2T
RECEIVES THE REQUEST AND SENDS THE RESPONSE
DOES NOT RECEIVE RESEND AND TIMER= 4T
RECEIVES THE REQUEST SENDS THE RESPONSE
STILL DOES NOT RECEIVE RESENDS THE REQUEST AND TIMER =8T
RECEIVES THE REQUEST SENDS THE RESPONSE
CONNECTION FAILURE

Flow chart:

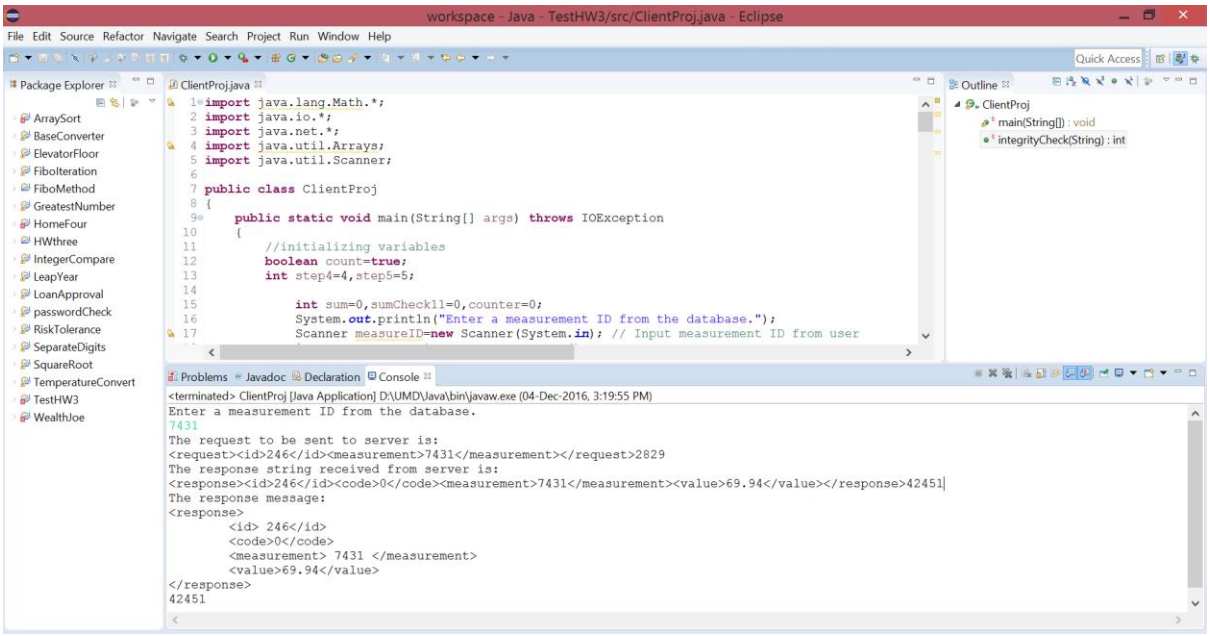

STATE TRANSITION DIAGRAM
CLIENT
SERVER
SENDS REQUEST AND STARTS TIMER, COUNTER=0
RECEIVES REQ AND SENDS RESPONSE.
RECEIVES RESP, RESETS TIMER, SENDS ANOTHER REQ
DOES NOT RECEIVE. RESEND, 2T, COUNTER ++
RECEIVES REQ AND SENDS RESPONSE.
RECEIVES RESPONSE, RESETS TIMER, SENDS ANOTHER REQ
DOES NOT RECEIVE. RESEND, 2T COUNTER ++
RECEIVES REQ AND SENDS RESPONSE.
DOES NOT RECEIVE. RESEND, 4T COUNTER+1
RECEIVES REQ AND SENDS RESPONSE.
DOES NOT RECEIVE RESEND, 8T, COUNTER++
RECEIVES REQ AND SENDS RESPONSE.
CONNECTION FAILURE


CLIENT SIDE
START
STEP 1 : SELECT MEASUREMENT ID, GENERATE REQ ID, COUNTER=0
STEP 2: ASSEMBLE AND SEND THE MESSAGE, REQ ID, START TIMER
RESEND, 2T
RESP RECEIVED BEFORE TIMEOUT
NO
COUNTER ++
COUNTER 4
YES
FAILURE
CHECK THE INTEGRITY OF THE MESSAGE & RESET TIMER
EQUAL?
NO
GO BACK TO STEP 2
YES
CHECK ERROR CODE
NO
DISPLAY ERROR MSG.
CODE=0
NO
CODE =1?
YES
ASK USER TO RESEND
YES
DISPLAY TEMP.
GO BACK TO STEP 1 UNTIL TERMINATION

Output:

- In case of correct response being received
  Output at the client:

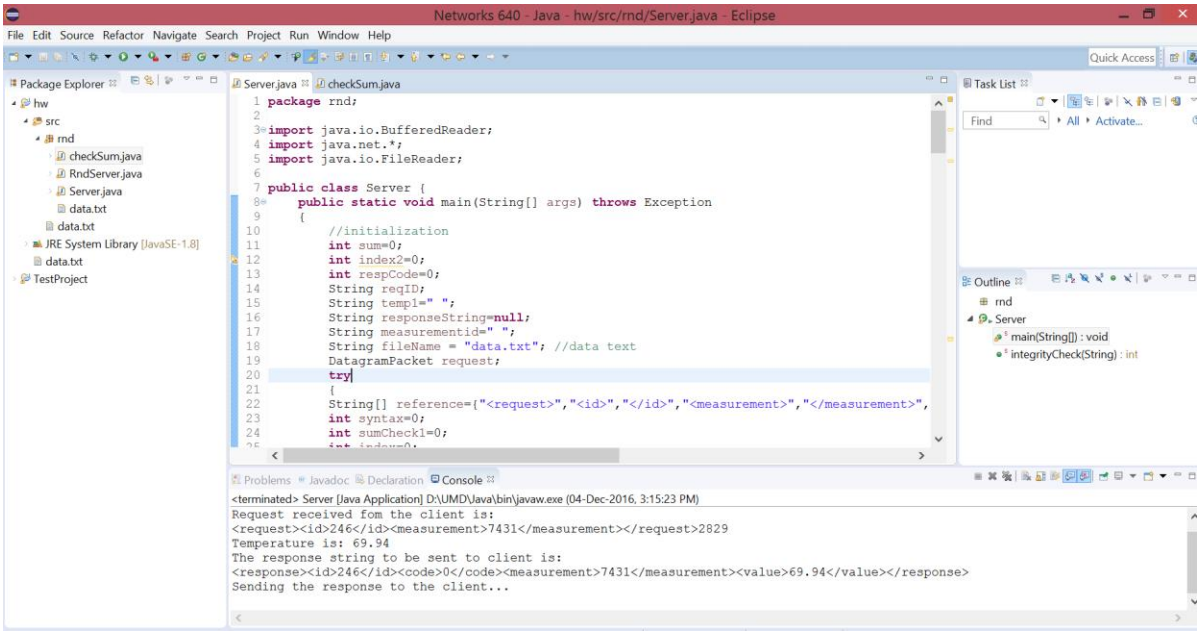
```
Enter a measurement ID from the database.
7431
The request to be sent to server is:
<request><id>246</id><measurement>7431</measurement><
/request>2829
The response string received from server is:
<response><id>246</id><code>0</code><measurement>7431
</measurement><value>69.94</value></response>42451
The response message:
<response>
      <id> 246</id>
      <code>0</code>
      <measurement> 7431 </measurement>
      <value>69.94</value>
</response>
42451
```



Output at the server:

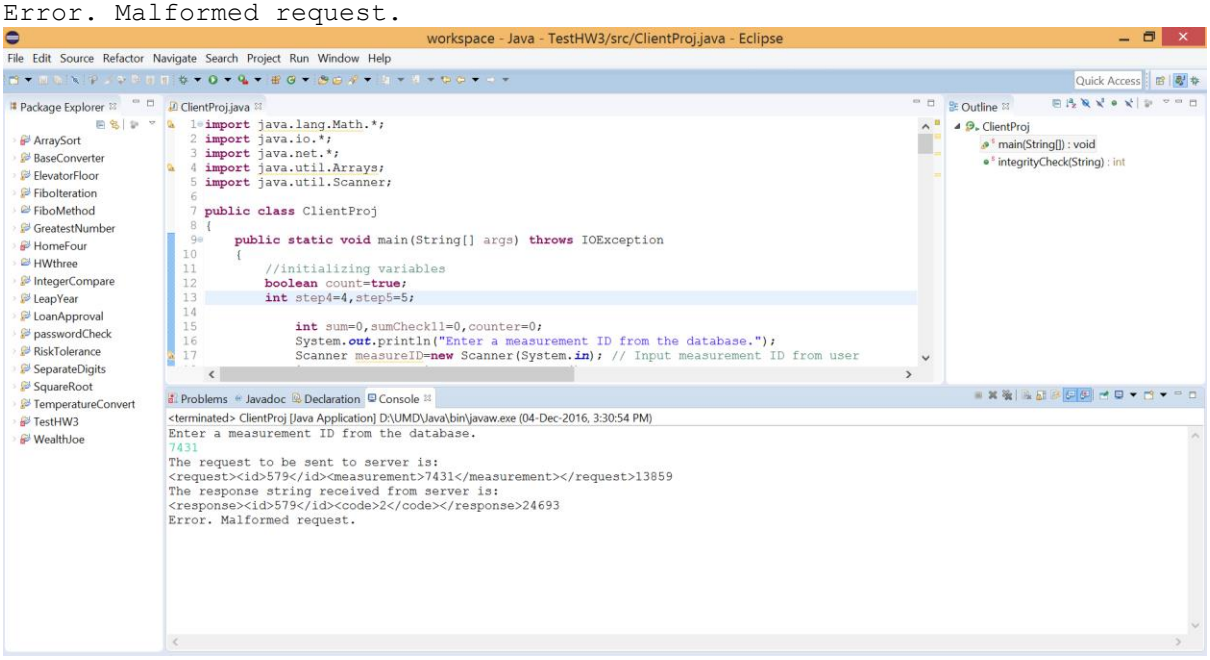
```
Receiving the request from the client...
Request received fom the client is:
<request><id>246</id><measurement>7431</measurement><
/request>2829
Temperature is: 69.94
The response string to be sent to client is:
<response><id>246</id><code>0</code><measurement>7431
</measurement><value>69.94</value></response>
```

```
Sending the response to the client...
```



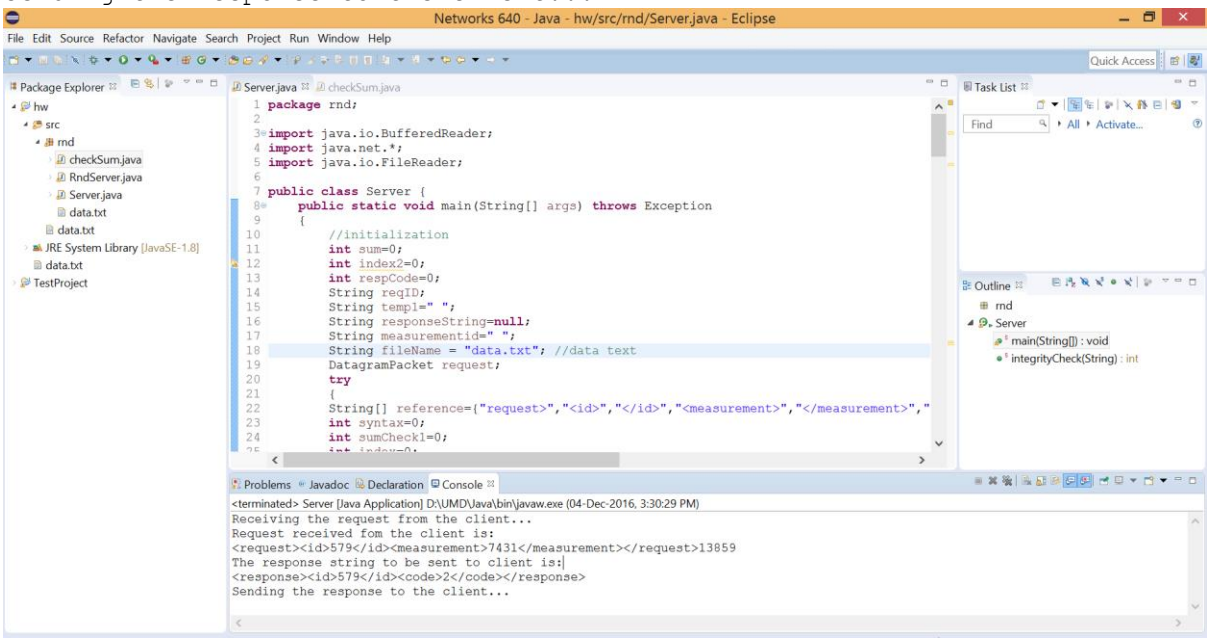
- In case of wrong syntax
  Output at the client

```
Enter a measurement ID from the database.
7431
The request to be sent to server is:
<request><id>579</id><measurement>7431</measurement><
/request>13859
The response string received from server is:
<response><id>579</id><code>2</code></response>24693
Error. Malformed request.
```



Output at the server

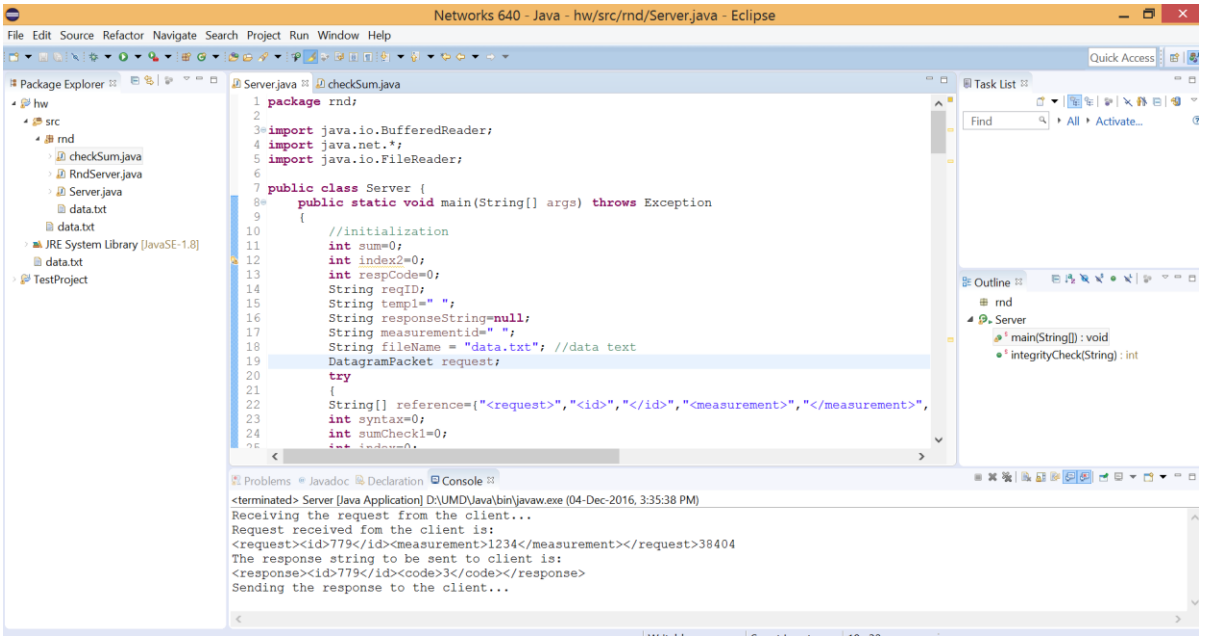
```
Receiving the request from the client...
Request received fom the client is:
<request><id>579</id><measurement>7431</measurement><
/request>13859
The response string to be sent to client is:
<response><id>579</id><code>2</code></response>
Sending the response to the client...
```



- **In case of wrong measurement ID**

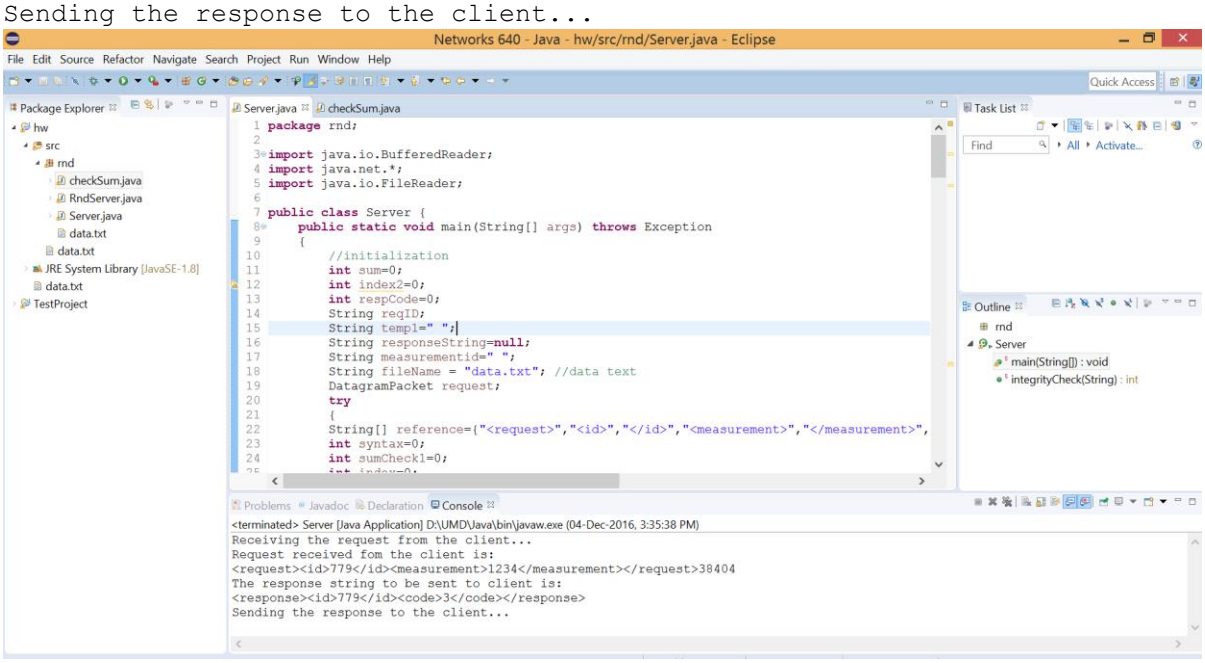
  Output at the client

```
Enter a measurement ID from the database.
1234
The request to be sent to server is:
<request><id>779</id><measurement>1234</measurement><
/request>38404
The response string received from server is:
<response><id>779</id><code>3</code></response>25461
Error. The measurement ID is invalid.
```



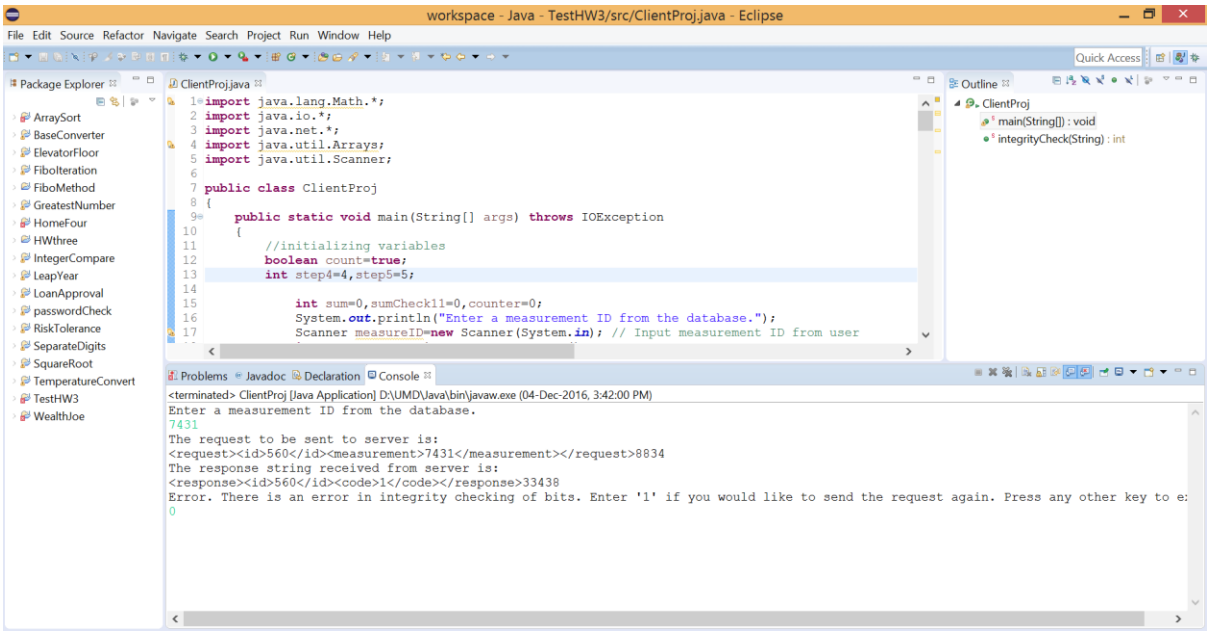
Output at the server

```
Receiving the request from the client...
Request received fom the client is:
<request><id>779</id><measurement>1234</measurement><
/request>38404
The response string to be sent to client is:
<response><id>779</id><code>3</code></response>
Sending the response to the client...
```



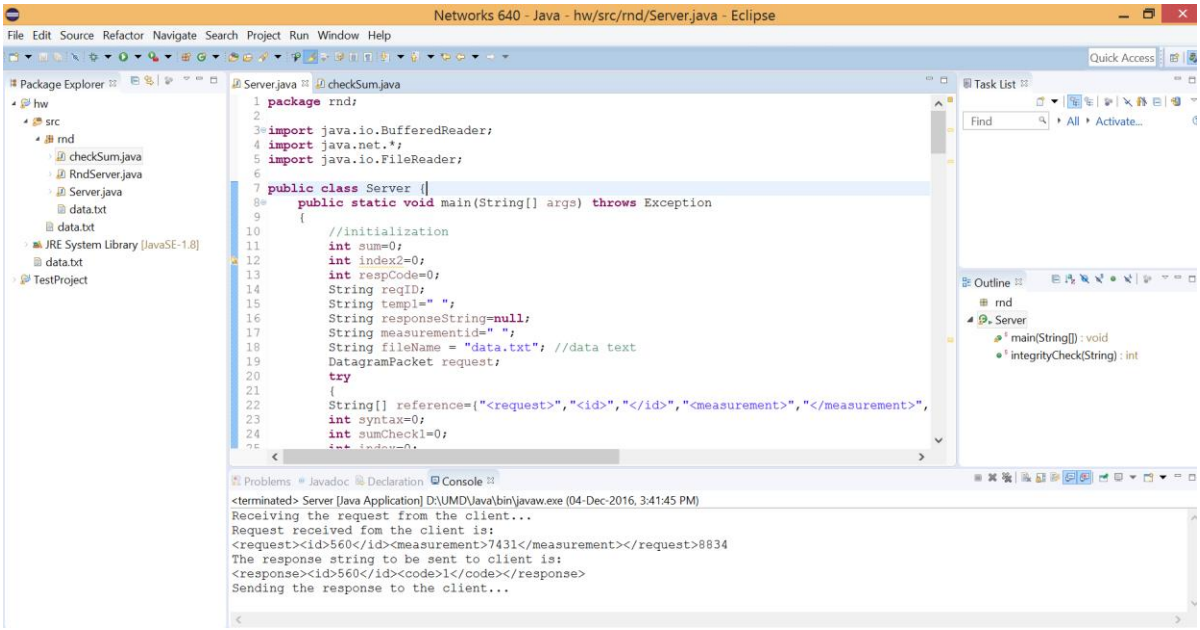
- **In case of checksum error and generation of response code 1**
  Output at the client:

```
Enter a measurement ID from the database.
7431
The request to be sent to server is:
<request><id>560</id><measurement>7431</measurement><
/request>8834
The response string received from server is:
<response><id>560</id><code>1</code></response>33438
Error. There is an error in integrity checking of
bits. Enter '1' if you would like to send the request
again. Press any other key to exit.
0
```



Output at the server:

```
Receiving the request from the client...
Request received fom the client is:
<request><id>560</id><measurement>7431</measurement><
/request>8834
The response string to be sent to client is:
<response><id>560</id><code>1</code></response>
Sending the response to the client...
```


Networks 640 - Java - hw/src/rnd/Server.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Package Explorer
hw
src
rnd
checkSum.java
RndServer.java
Server.java
data.txt
data.txt
JRE System Library [JavaSE-1.8]
data.txt
TestProject
Server.java
checkSum.java
1 package rnd;
2
3 import java.io.BufferedReader;
4 import java.net.*;
5 import java.io.FileReader;
6
7 public class Server {
8 public static void main(String[] args) throws Exception
9 {
10 //initialization
11 int sum=0;
12 int index2=0;
13 int respCode=0;
14 String reqID;
15 String temp1=" ";
16 String responseString=null;
17 String measurementid=" ";
18 String fileName = "data.txt"; //data text
19 DatagramPacket request;
20 try
21 {
22 String[] reference={"<request>","<id>","</id>","<measurement>","</measurement>",
23 int syntax=0;
24 int sumCheck1=0;
Task List
Find
All
Activate...
Outline
rnd
Server
main(String[]) : void
integrityCheck(String) : int
Problems
Javadoc
Declaration
Console
<terminated> Server [Java Application] D:\UMD\Java\bin\javaw.exe (04-Dec-2016, 3:41:45 PM)
Receiving the request from the client...
Request received fom the client is:
<request><id>560</id><measurement>7431</measurement></request>8834
The response string to be sent to client is:
<response><id>560</id><code>1</code></response>
Sending the response to the client...

Challenges and difficulties faced during the entire process

Client side

Timer conditions

Problem

After the timer expires ,we had to go to the catch block and resend the request ,increment the timer and after receiving the message reset the timer.The client side was however not receiving the message.Though the server sends it.It entered the try block but the socket connection was giving an error.

Solution

Adjusting the socket parameters using the socket functions so that any instantaneous change does not affect the smooth working of the process.

Server side

Integrity checksum

Problem

After transmitting the message from the client side ,when we display the message at the server side ,the message after checksum part had a lot varying unknown characters attached to it.As a result the new checksum did not match the original message though the message was correct.

Solution

As the only solution to this problem was to get rid of those unknown characters,after receiving the message at the server side we tried to convert the entire sting into its ascii characters to know of these errors,but it did not provide the solution.We also tried to resolve it using the length of the byte array transmitted. At the end , trim function was used to get the correct checksum calculation.

OUTPUTS

Output:

- In case of correct response being received

Output at the client:

```
Enter a measurement ID from the database.

7431

The request to be sent to server is:

<request><id>246</id><measurement>7431</measureme
nt></request>2829

The response string received from server is:

<response><id>246</id><code>0</code><measurement>
7431</measurement><value>69.94</value></response>
42451

The response message:

<response>

      <id> 246</id>

      <code>0</code>

      <measurement> 7431 </measurement>

      <value>69.94</value>

</response>

42451
```

Output at the server:

```
Receiving the request from the client...

Request received fom the client is:

<request><id>246</id><measurement>7431</measureme
nt></request>2829

Temperature is: 69.94

The response string to be sent to client is:
```

<response><id>246</id><code>0</code><measurement>
7431</measurement><value>69.94</value></response>

Sending the response to the client...

- In case of wrong syntax

Output at the client

Enter a measurement ID from the database.

7431

The request to be sent to server is:

<request><id>579</id><measurement>7431</measureme
nt></request>13859

The response string received from server is:

<response><id>579</id><code>2</code></response>24
693

Error. Malformed request.

Output at the server

Receiving the request from the client...

Request received fom the client is:

<request><id>579</id><measurement>7431</measureme
nt></request>13859

The response string to be sent to client is:

<response><id>579</id><code>2</code></response>

Sending the response to the client...

- **In case of wrong measurement ID**

Output at the client

```
Enter a measurement ID from the database.
1234
The request to be sent to server is:
<request><id>779</id><measurement>1234</measureme
nt></request>38404
The response string received from server is:
<response><id>779</id><code>3</code></response>25
461
Error. The measurement ID is invalid.
```

Output at the server

```
Receiving the request from the client...
Request received fom the client is:
<request><id>779</id><measurement>1234</measureme
nt></request>38404
The response string to be sent to client is:
<response><id>779</id><code>3</code></response>
Sending the response to the client...
```

- **In case of checksum error and generation of response code 1**

Output at the client:

```
Enter a measurement ID from the database.
```

7431

The request to be sent to server is:

<request><id>560</id><measurement>7431</measurement></request>8834

The response string received from server is:

<response><id>560</id><code>1</code></response>33438

Error. There is an error in integrity checking of bits. Enter '1' if you would like to send the request again. Press any other key to exit.

0

Output at the server:

Receiving the request from the client...

Request received fom the client is:

<request><id>560</id><measurement>7431</measurement></request>8834

The response string to be sent to client is:

<response><id>560</id><code>1</code></response>

Sending the response to the client...

- **Time out condition**

Output at client

Output at server