

# PROJECT REPORT

ENTS 640: NETWORKS AND PROTOCOLS I

FALL 2016

"I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination."

Submitted by:

Ishan Chauhan UID: 114859051

Rishiraj Charan UID: 115016842

# TABLE OF CONTENTS

Serial Number	Topic	Page Number
1	Problem Description	3
2	Solution Approach	
	A) Flow Diagrams	5
	B)UML Class Diagrams and Main Class Design	8
3	Applications Output	10

## PROBLEM DESCRIPTION:

Standard UDP is an unreliable connection less protocol employed for loss tolerant, low latency applications. It is packet oriented and connection less which means that connection state information is never maintained and no connection setup/ teardown are required. The problem attempts to improve the reliability of UDP services using an application that is using UDP datagrams by adding extra features like integrity checksum, timeouts & retransmissions.

The problem in the project assignment is to design a client server communication application in which the client requests a temperature value from the server for a measurement id. The measurement id and temperature values are both available in a text file on the system. The server is to respond with the respective temperature value for that measurement id. If the client does not receive the response within 1 second i.e. initial timeout value, the client should resend the request and double the timeout value. This process of timeout & retransmission can continue till 4th timeout after which the application is to be terminated. The timeout & retransmission features improve the reliability of this application significantly by ensuring that the client gets all the requested responses and in the same order as were required.

Request & response messages between the client and the server are to be in human readable text, also a defined format is to be followed in the messages. For example, the entire request message excluding integrity checksum is to be enclosed between the tags "<request>" & "</request>", similarly all the message elements are to be enclosed within their respective tags. The checksum is to be calculated over the entire message up till the last closing tag '>'. The calculated integrity checksum is to be appended to this message before transmission. The same procedure is to be followed for the response message as well. The integrity checksum value ensures that the data is not corrupted during transmission. Before transmission of any of the messages, string messages are to be converted to an array of bytes as is the need for using UDP datagram packets in JAVA.

Upon successful reception of the message by server, it first calculates the integrity checksum of the message and then compares it with the one provided in the message. If they were to be equal, it shall proceed further and if not it shall assemble a response message with response code as 1 which indicates that the message was corrupted. If the comparison turns out to be true, the server shall convert the byte array of message into human readable string message object. Then it shall check the syntax of request message which includes closing/opening tags, their order, misspellings and will also check if the measurement id is 16-bit unsigned integer. If the syntax is wrong, it shall assemble a response message with the request id and the response code set to 2 indicating invalid syntax. If the syntax is right the server shall lookup the measurement value with the ones available in the data text file. If there is none, it shall assemble a response message with request id and response code set to 3 indicating invalid measurement value. If the measurement value is present, it shall extract the corresponding temperature value. It shall assemble the response message with request id, measurement value, response code set to 0 and temperature value. This process shall be repeated using a loop until it is manually terminated.

Client shall wait for the response message with a timer, at each timeout the timeout value is doubled and this shall continue until 4th timeout after which the application is terminated. Upon reception, client shall calculate the checksum and compare it with the one in response message.

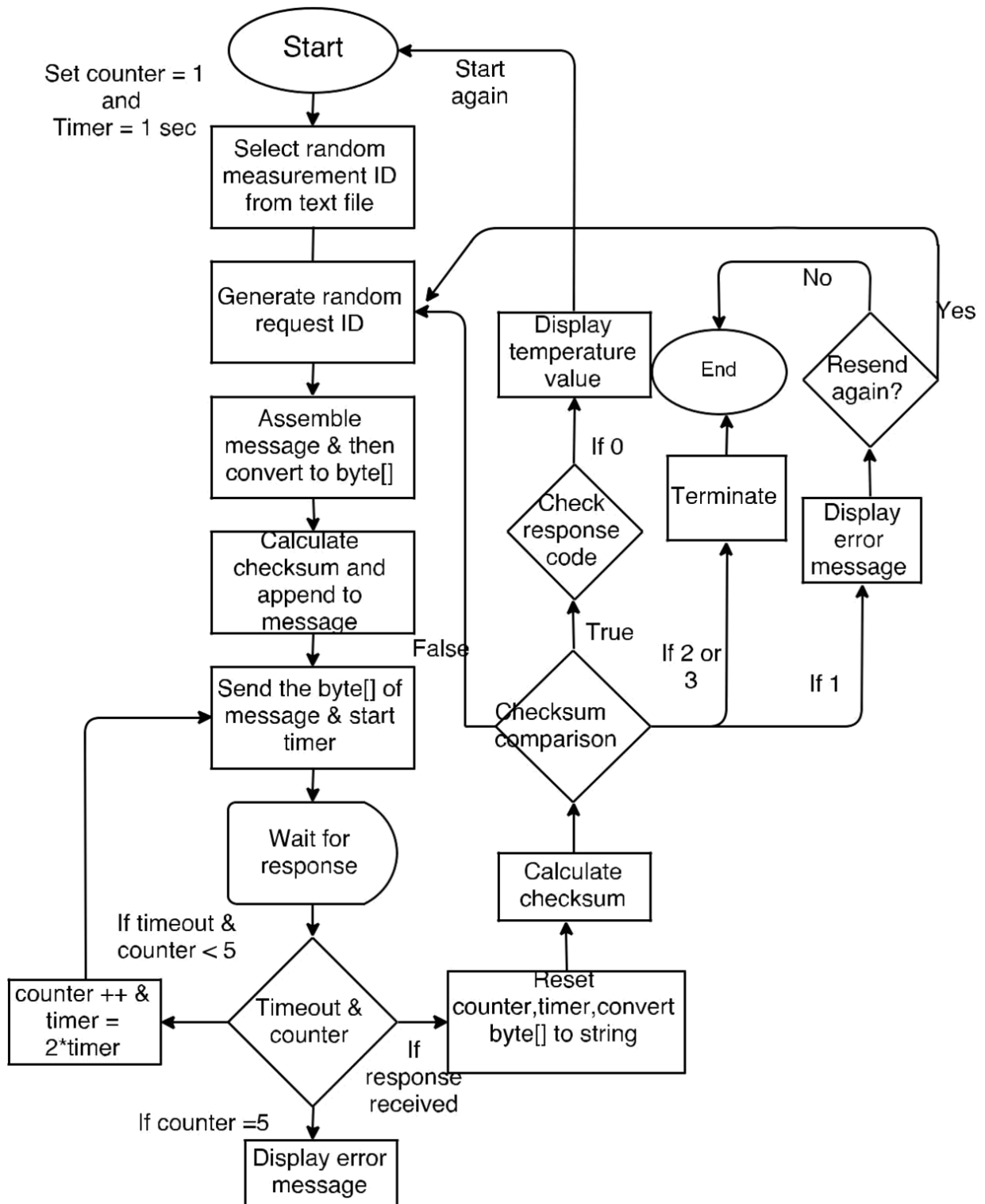
If the comparison fails, it shall continue to resend and receive in loop. If comparison is true, it shall extract its response code. If code is anything other than 0, client shall display an error message, loop and resend the same measurement request. In addition, if the code is 1, it shall ask user to resend the same request or not. It shall display the temperature value in case when code is 0 and then start the whole process to send a new measurement request.

To calculate integrity checksum, the ASCII code for each character shall be concatenated with one consecutive one succeeding it to form a 16-bit unsigned integer. If character sequence is odd, the last character sequence shall be assumed to be 0. This way an array of 16 bit unsigned integers shall be created. For each word index, shall be calculated by an exclusive or between checksum- S (initialized to 0) and the word. Next the checksum variable S will be assigned a value by multiplying index with a constant  $C = 7919$  and then doing a modulus operation with constant  $D = 65536$ . Same is repeated for each word of the array to obtain the checksum value in variable S.

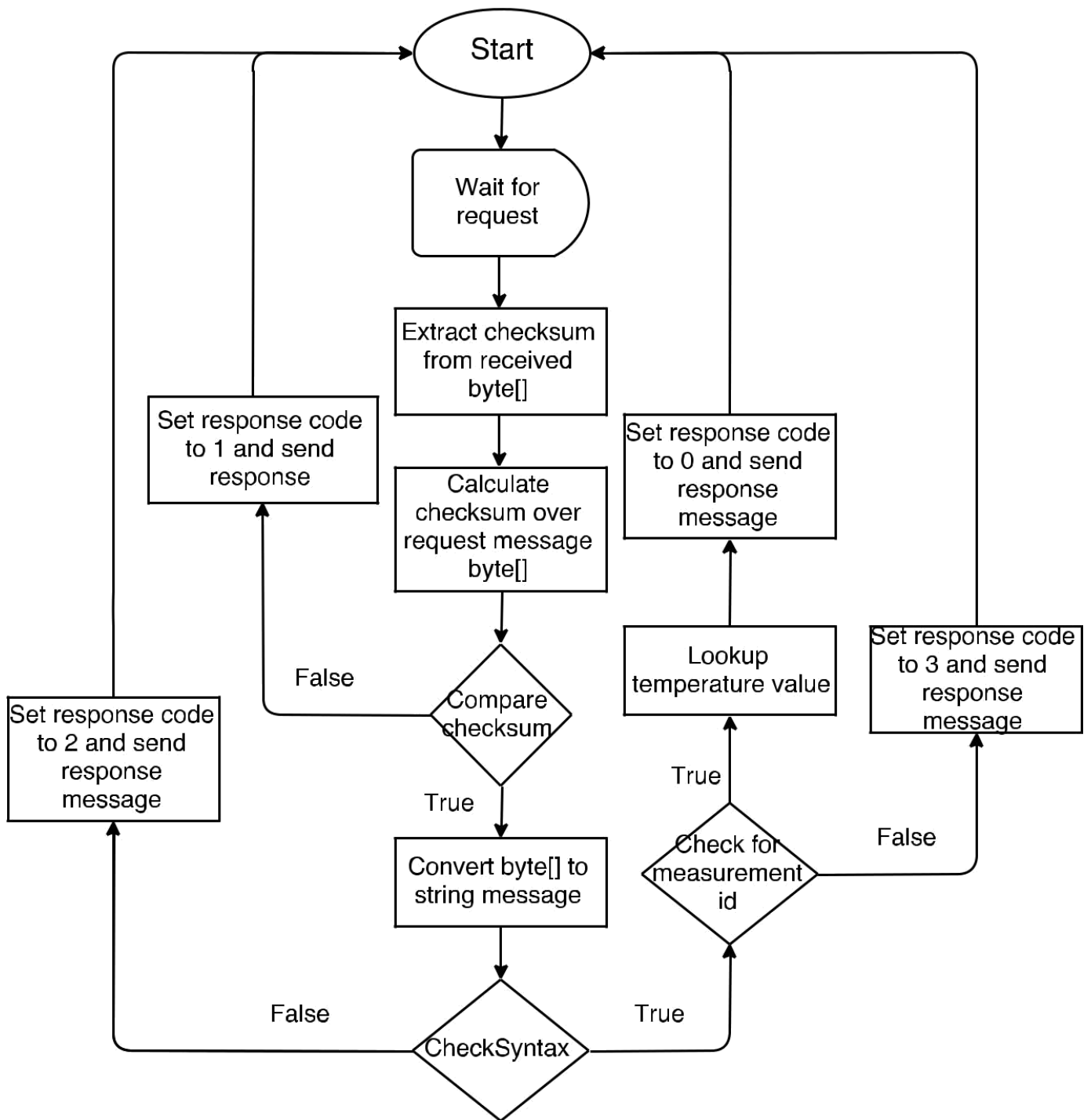
# SOLUTION APPROACH:

## A.) FLOW DIAGRAMS:

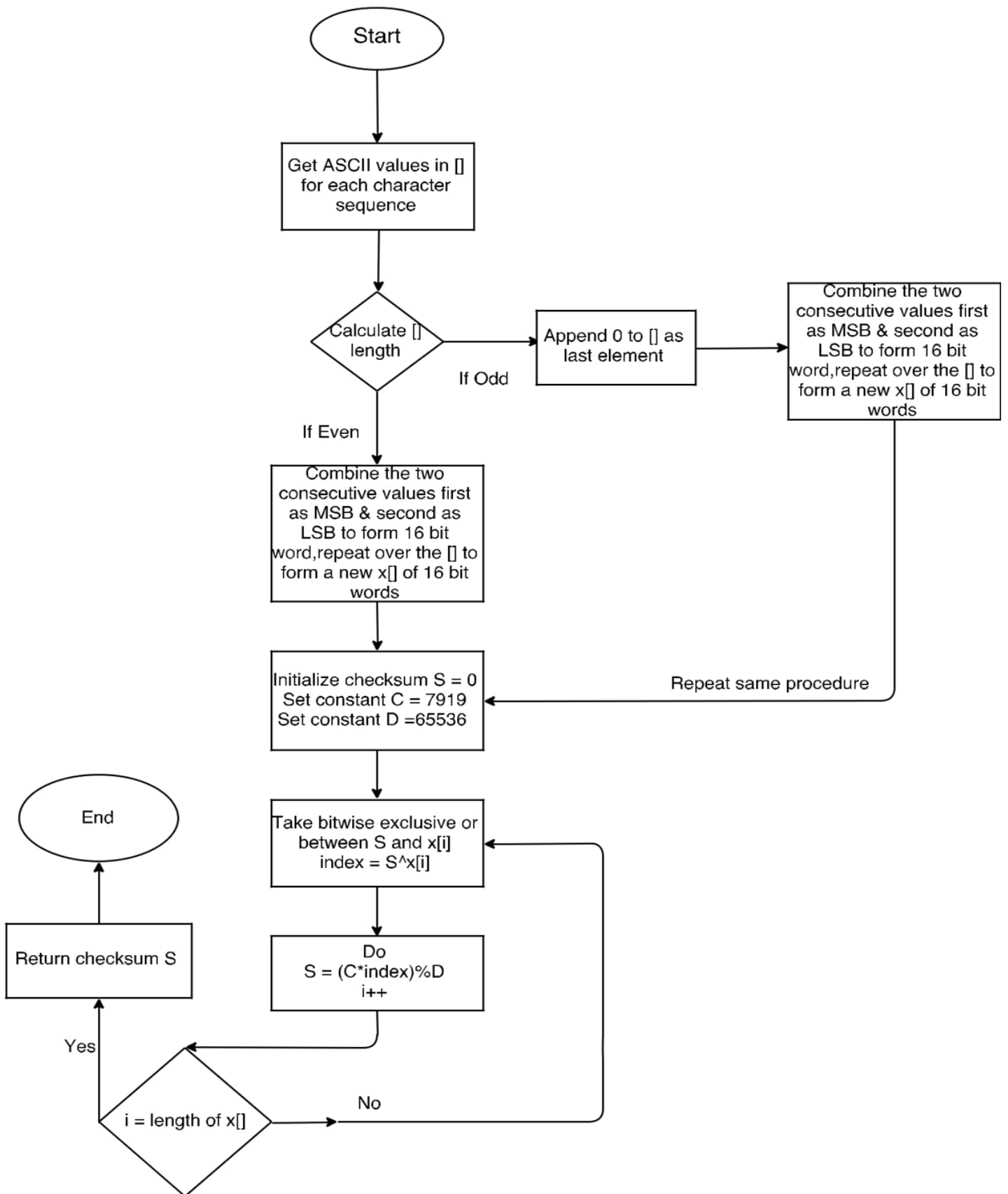
Client Side



Server Side



# Checksum Calculation Flow Diagram

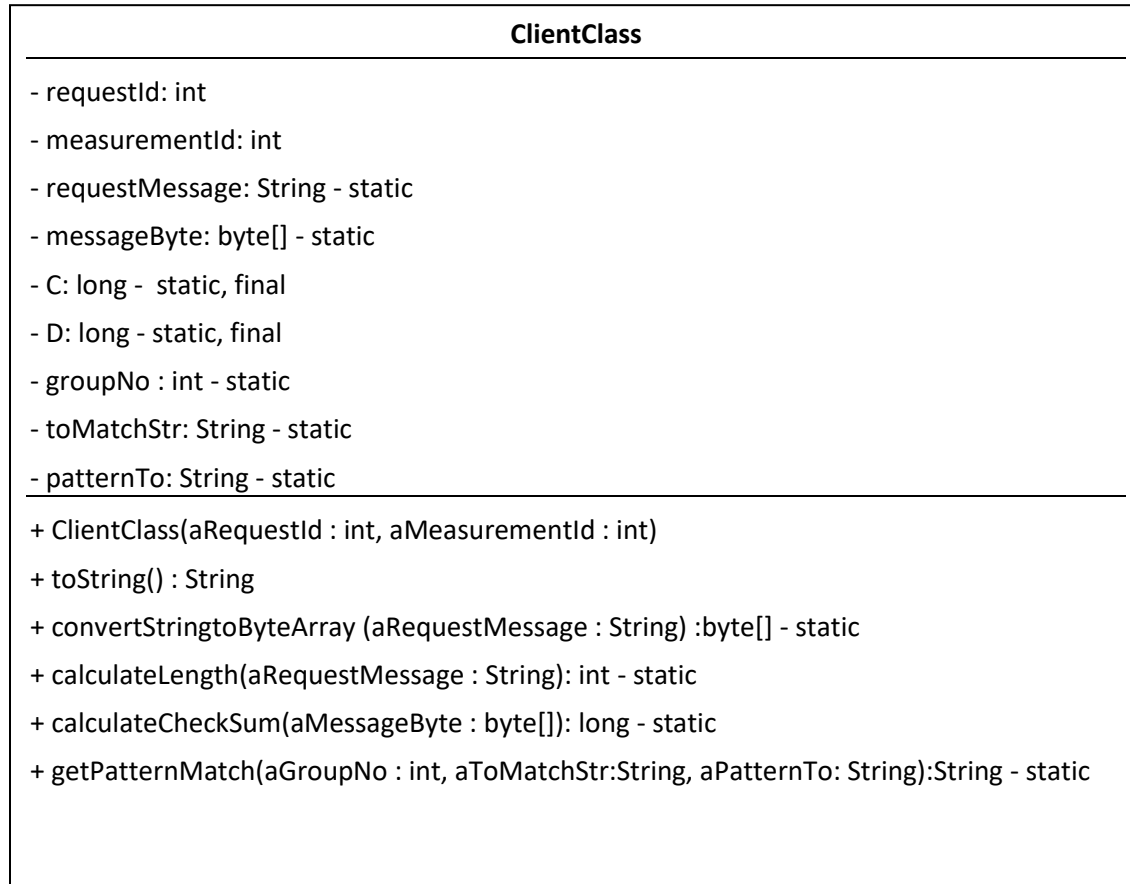


## B.) UML CLASS DIAGRAM AND MAIN CLASS DESIGN

For both client & server, we have one main tester class and one method class to implement certain methods that are used repeatedly and to avoid duplicity.

We have implemented mostly static methods since we did not need any particular implementation of base class/object and static methods gave us more flexibility in avoiding duplicate codes.

Client side:



Constants C & D are used to calculate checksum, groupNo , toMatchStr & patternTo are used in String regular expressions .

ClientClass() is a constructor method, toString() method returns the request message for an object created using request id & measurement id.

ConvertStringtoByteArray() method returns a byte array for a message, calculateLengt() method returns length of a string.

CalculateChecksum() method inputs the byte[] of a message and returns the appropriate checksum.

GetPatternMatch() method is a method which uses regular expressions and returns the required string for a pattern in a string.

The rest of the functions/steps have been implemented in the main class.



ServerClass
<ul style="list-style-type: none"> <li>- byteArray: byte[]</li> <li>- msgString: String - static</li> <li>- id: String - static</li> <li>- code: int - static</li> <li>- measurement: String - static</li> <li>- value: String - static</li> <li>- C: long - static, final</li> <li>- D: long - static, final</li> <li>- groupNo: int - static</li> <li>- measurId: long - static</li> <li>- toMatchStr: String - static</li> <li>- patternTo: String - static</li> <li>- responseMessage: String - static</li> </ul>
<ul style="list-style-type: none"> <li>+ ServerClass(abytArray: byte[])</li> <li>+ calculateChecksum(abytArray : byte[]): long</li> <li>+ checkSyntax(aMsgString: String, aMeasurId: long): Boolean - static</li> <li>+ createMessage(ald: String, aCode: int, aMeasurement: String, aValue: String) :String - static</li> <li>+ convertStringtoByteArray(aResponseMessage: String): byte[] - static</li> <li>+ getPatternMatch(aGroupNo : int, aToMatchStr:String, aPatternTo: String):String – static</li> </ul>

Constants C & D are used to calculate checksum, groupNo , toMatchStr & patternTo are used in String regular expressions .

CalculateChecksum() method inputs the byte[] of a message and returns the appropriate checksum.

GetPatternMatch() method is a method which uses regular expressions and returns the required string for a pattern in a string.

ConvertStringtoByteArray() method returns a byte array for a message.

checkSyntax() method checks the syntax of the received message which includes checking all opening & closing tags with appropriate labels, checking the order of all labels, checking whether measurement id is a valid unsigned 16 bit integer and finally ensuring that it does not contain any invalid character. It returns true if all conditions are met.

createMessage() method creates the response message after supplying it with appropriate information.

The rest of the functions/steps have been implemented in the main class.

## APPLICATION'S OUTPUT:

- In normal operation without any errors i.e. response code is 0

Client side:

```
<terminated> ClientMain [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe
<request>
  <id>23253</id>
  <measurement>16047</measurement>
</request>
37494

Receiving response from server: Attempt #1
The measurement value of 16047 is :97.62
```

Server side:

```
<terminated> ServerMain [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe
<response>
  <id>23253</id>
  <code>0</code>
  <measurement>16047</measurement>
  <value>97.62</value>
</response>54348
```

In a loop- client side:

```
<terminated> ClientMain [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe
<request>
  <id>23253</id>
  <measurement>16047</measurement>
</request>
37494

Receiving response from server: Attempt #1
The measurement value of 16047 is :97.62

<request>
  <id>15957</id>
  <measurement>20489</measurement>
</request>
27802

Receiving response from server: Attempt #1
The measurement value of 20489 is :81.19

<request>
  <id>47122</id>
  <measurement>10624</measurement>
</request>
23968
```

In a loop – server side:

```
<terminated> ServerMain [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\
<response>
  <id>23253</id>
  <code>0</code>
  <measurement>16047</measurement>
  <value>97.62</value>
</response>54348

<response>
  <id>15957</id>
  <code>0</code>
  <measurement>20489</measurement>
  <value>81.19</value>
</response>14253

<response>
  <id>47122</id>
  <code>0</code>
  <measurement>10624</measurement>
  <value>67.27</value>
</response>27954
```

- When there is communication failure (achieved by not running server side)- client side:

```
<terminated> ClientMain [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (
|request>
  <id>7767</id>
  <measurement>40534</measurement>
</request>
25052

Receiving response from server: Attempt #1
Receiving response from server: Attempt #2
Receiving response from server: Attempt #3
Receiving response from server: Attempt #4
ERROR: Communication failure
```

- When the checksum calculated at client side is not same as the one supplied in response message, client loops from step 2 to resend same measurement id request.

```
<terminated> ClientMain [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\j
<request>
  <id>1254</id>
  <measurement>12704</measurement>
</request>
36638

Receiving response from server: Attempt #1
<request>
  <id>20468</id>
  <measurement>12704</measurement>
</request>
53741

Receiving response from server: Attempt #1
<request>
  <id>60769</id>
  <measurement>12704</measurement>
</request>
33473
```

- When syntax of request message is wrong i.e. response code is 2(in this case a tag is missing). Client terminates the application when code is either 2 or 3.

Client side:

```
<terminated> ClientMain [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (Dec .
<request>
    62908</id>
    <measurement>58071</measurement>
</request>
8073

Error: malformed request. The syntax of the request message is not correct
Receiving response from server: Attempt #1
```

- When measurement id is not valid i.e. response code is 3. Client terminates the application when code is either 2 or 3.

```
<terminated> ClientMain [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (Dec 2, 2016, 11:58:05 AM)
<request>
    <id>50208</id>
    <measurement>1234</measurement>
</request>
26023

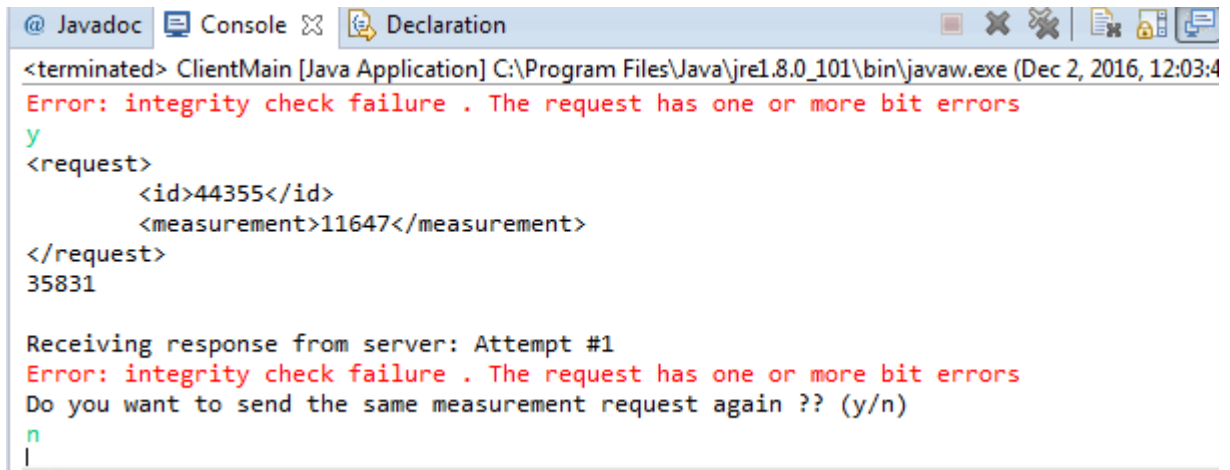
Receiving response from server: Attempt #1
Error: non-existent measurement. The measurement with the requested measurement ID does not exist.
```

- When integrity check fails at the server side i.e. when response code is 1. Client application may go to step 2 to resend same measurement request again based on user choice(y/n).

```
<terminated> ClientMain [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (Dec 2, 2016, 1
<request>
    <id>11153</id>
    <measurement>11647</measurement>
</request>
52946

Receiving response from server: Attempt #1
Do you want to send the same measurement request again ?? (y/n)
Error: integrity check failure . The request has one or more bit errors
y
<request>
    <id>44355</id>
    <measurement>11647</measurement>
</request>
35831
```

When choice is “n”, application terminates



```
@ Javadoc Console Declaration
<terminated> ClientMain [Java Application] C:\Program Files\Java\jre1.8.0_101\bin\javaw.exe (Dec 2, 2016, 12:03:4
Error: integrity check failure . The request has one or more bit errors
y
<request>
  <id>44355</id>
  <measurement>11647</measurement>
</request>
35831

Receiving response from server: Attempt #1
Error: integrity check failure . The request has one or more bit errors
Do you want to send the same measurement request again ?? (y/n)
n
|
```

-----X-----X-----X-----