ENTS 640 Networks and Protocols I

Fall 2016

Project Report

Ashish Joshi

Yaddel Ezeribe

Contents

[Introduction 3](#_Toc468456491)

[UDP Overview 3](#_Toc468456492)

[Project Overview 3](#_Toc468456493)

[UML Diagrams and Class description 3](#_Toc468456494)

[Client Sequence Diagram 3](#_Toc468456495)

[Server Sequence Diagram 4](#_Toc468456496)

[Common Classes 4](#_Toc468456497)

[Data Lookup 4](#_Toc468456498)

[Integrity Check 5](#_Toc468456499)

[Client Classes and Description 6](#_Toc468456500)

[Error Client Main / Client Main 6](#_Toc468456501)

[Client Request Message 6](#_Toc468456502)

[Decode Server Response 7](#_Toc468456503)

[Server Classes and Description 8](#_Toc468456504)

[Server Main 8](#_Toc468456505)

[Decode Client Message 8](#_Toc468456506)

[Flow Chart 10](#_Toc468456507)

[Client Side Operation Flow 10](#_Toc468456508)

[Server Side Operation Flow 11](#_Toc468456509)

[Algorithm 12](#_Toc468456510)

[Client Request 12](#_Toc468456511)

[Server Receive and Response 12](#_Toc468456512)

[Client Receive 12](#_Toc468456513)

[Test Cases and output 13](#_Toc468456514)

[General output without error 13](#_Toc468456515)

[Integrity Check 13](#_Toc468456516)

[Malformed Request 14](#_Toc468456517)

[Non-existant Measurement 15](#_Toc468456518)

[Request Timeout 16](#_Toc468456519)

# Introduction

## UDP Overview

UDP (User Datagram Protocol) is a communications protocol used primarily for establishing low-latency and loss tolerant connections for data transfer. UDP is an alternative to TCP (Transmission Control Protocol) which is a connection oriented and reliable data transfer protocol. Both UDP and TCP run on transport layer on top of the Internet Protocol (Network Layer) and are sometimes referred to as UDP/IP or TCP/IP. The short data payload of transport layer is called datagrams, the term commonly used for both UDP and TCP.

UDP provides two services to the IP layer, first is port numbers to help distinguish different user requests and, second, a checksum capability to verify that the data integrity is intact.

TCP has been a dominant protocol used for Internet connectivity owing to its “reliable” services, but the reliability come at a cost in terms of additional data overhead, and latency. In contrast, UDP just sends the packets resulting in lower bandwidth overhead and latency. But during UDP transmission, packets can be lost or received out of order owing to the different paths individual datagram can take between sender and receiver.

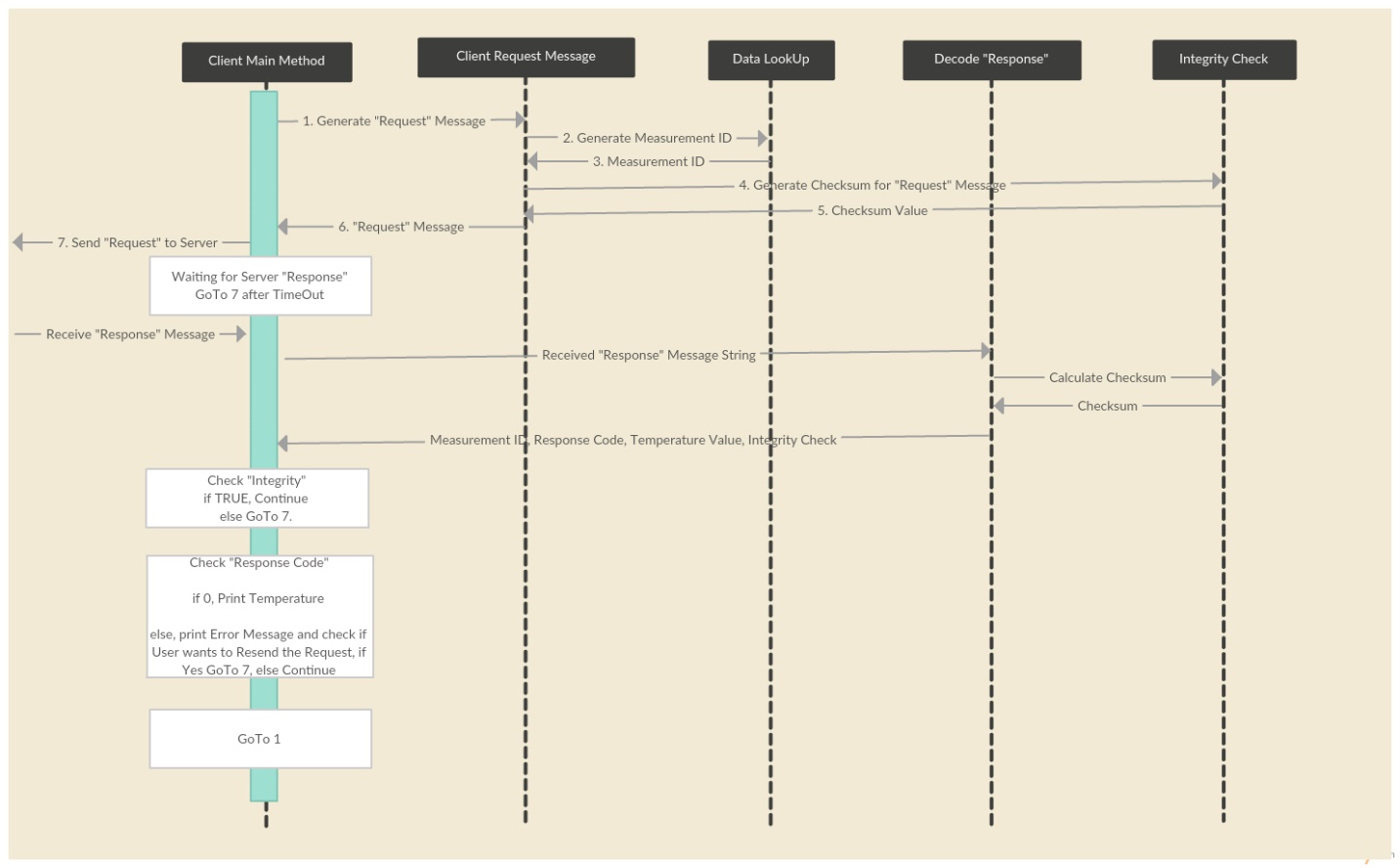
UDP is an ideal protocol for network applications in which perceived latency is critical such as gaming, voice and video communications. These can suffer some data loss without adversely affecting perceived quality. UDP works in conjunction with higher level protocols to help manage data transmission for Trivial File Transfer Protocol (TFTP), Real Time Streaming Protocol (RTSP), Simple Network Protocol (SNP) and Domain Name System (DNS) lookups.

## Project Overview

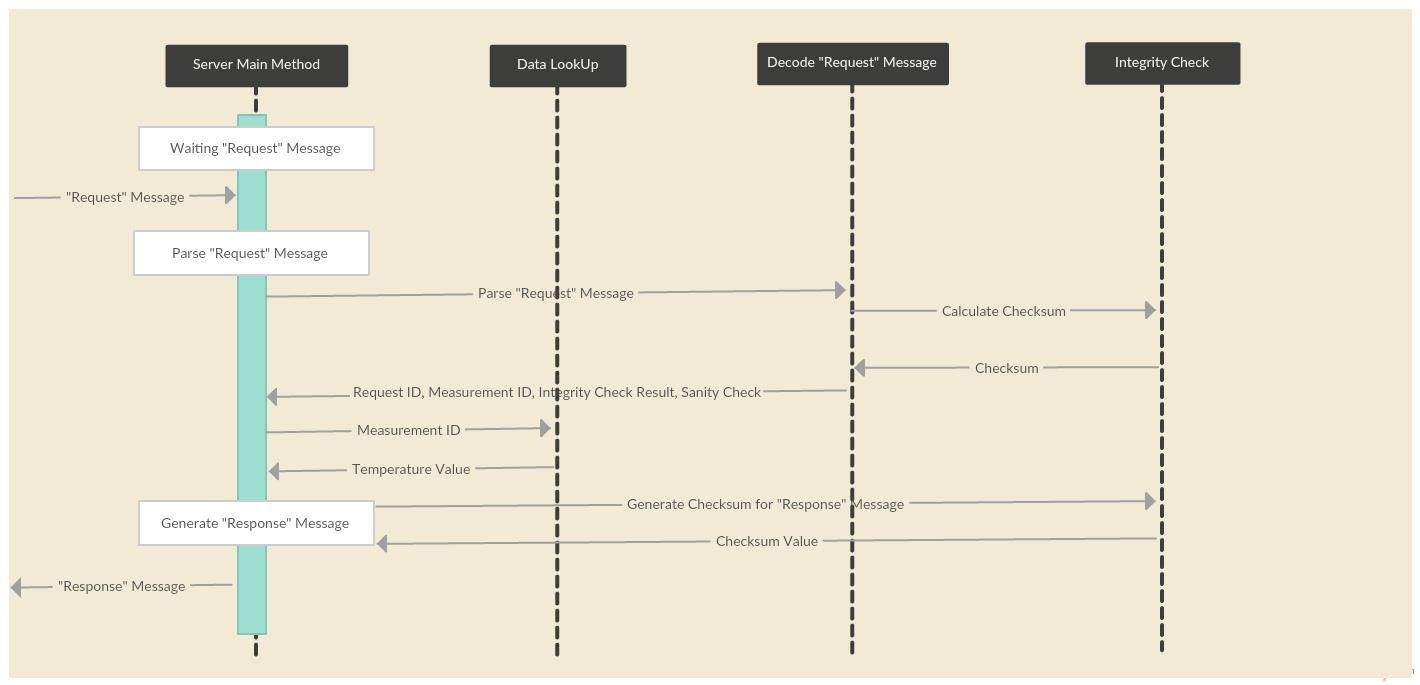
The program is designed to simulate communication between client and server using Java. Existing UDP functionalities of packet and socket is being used to send and receive datagrams. The client creates a request message specifying a measurement ID against which the server is supposed to respond with an associated Temperature value. The communication is continuous till it is terminated, it has features to handle anticipated errors and delay.

# UML Diagrams and Class description

## Client Sequence Diagram



## Server Sequence Diagram

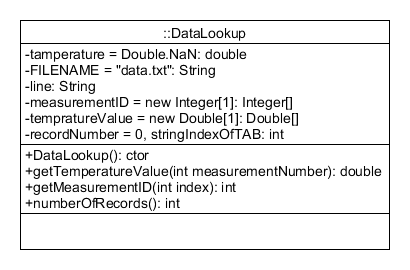


## Common Classes

There are two classes which are common in Client and Server.

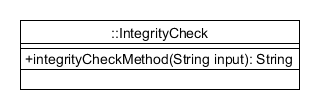
### Data Lookup

The Class process the data file “data.txt”. It accesses the file and read line by line till it reaches the end of file. The logic is to populate two arrays, one for Measurement ID and other for Temperature. While reading line by line, we are dividing the data set by identifying “white space” between the numbers. The relationship between both the arrays are maintained with the help of same “Index”. So for matching the corresponding values we first match Measurement ID, find out its Index, and then pull out the same Index value from Temperature array.



### Integrity Check

The class generates Checksum value for the set of bytes based on algorithm given in the project description. It takes a string as input, converts the String to array of bytes, group chunk 2 bytes into a short, and then computes checksum until it reaches end of array. The final checksum value is then converted to String and returned.



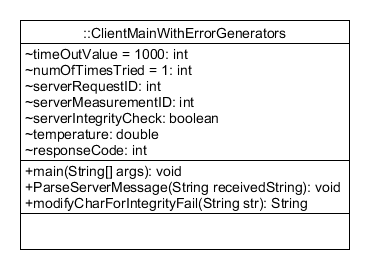
## Client Classes and Description

### Error Client Main / Client Main

Client Main class has the “main()” method, this is the class from where the program starts execution. This class is responsible for transmission-reception of messages, and display the output. There are two variations of this class in the project.

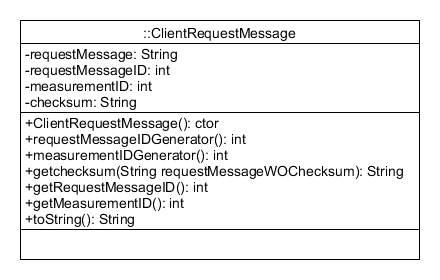
1. The first variant “Client Main” is for normal operation, which continuously keep transmitting and receiving the messages without user’s intervention. It keeps executing the code continuously and displays results till the user does not Terminates the application manually.
2. The second variant “Error Client Main” gives option to simulate Error scenarios to test the functionality of “Response Codes”.

The functionalities implemented in this class are transmission, retry in case of timeout, reception, and in case there is an error during operation display the error type and provide option to resend the message.



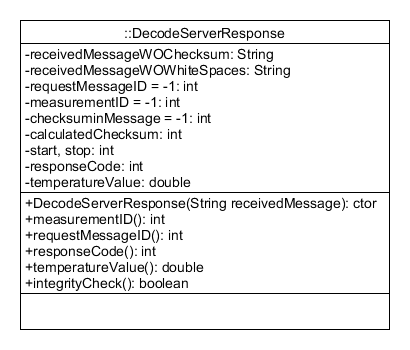
### Client Request Message

The class prepares a request message in string format and return the message string. It randomly generates Request ID and Measurement ID, arrange them in the format specified in the project description. It also calls the Integrity Check method to generate checksum. It then appends the computed checksum to the message string and returns the string to Client Main class.



### Decode Server Response

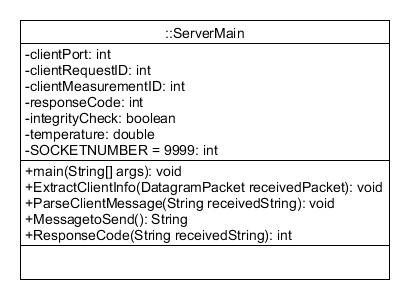
The Client Main class passes received message string to this class. The class decodes the received string and identify Request ID, Measurement ID, Response Code, and Temperature value. It also validates the Checksum and checks if Integrity Check passed or failed.



## Server Classes and Description

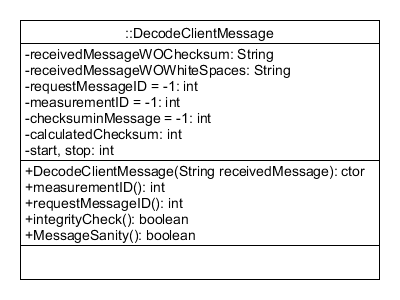
### Server Main

The class contains “main()” method, so this is the class from where the program execution starts. The class is responsible for reception of Request message and transmission of Response Message. The class calls Decode Client Message class to parse the Request message. To prepare the Response message the same Request ID and Measurement ID is used from the client Request message. In addition, it calculates Response Code, fetches Temperature value and checksum.



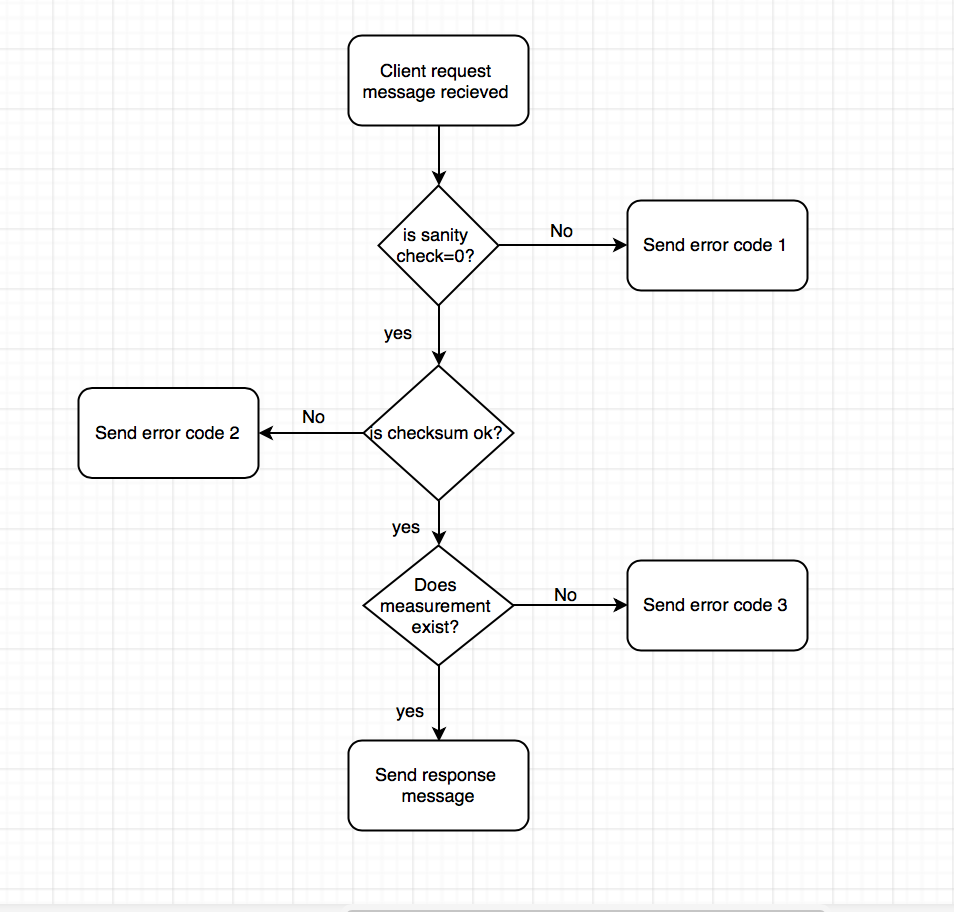
### Decode Client Message

The class decodes the string sent by Server Main class. First, it removes the any whitespace from the received message string. Then, it decodes the Request ID, Measurement ID, validates the checksum to check integrity of received message and does the sanity check to validate the Request message format. The format should be as given in the project description.

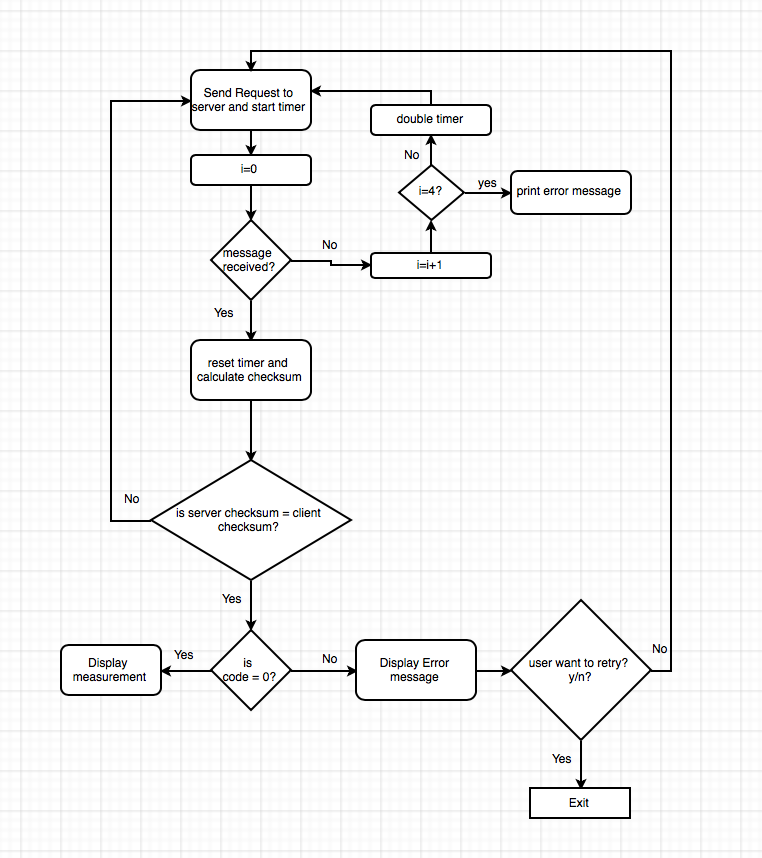


# Flow Chart

## Client Side Operation Flow



## Server Side Operation Flow



# Algorithm

## Client Request

1. Generate random Index number between (0-99).
2. Go to that Index in Measurement ID Array, fetch the Measurement ID.
3. Generate positive Random number less than 65536 and use that as Request ID.
4. Create Request Message String using Request Message format, put the Request ID and Measurement ID as obtained from Step 2 & 3.
5. Calculate Checksum for Request Message String obtained from Step 4.
6. Modify Request Message String to append the calculated Checksum (from Step 5) at the end
7. Convert the Request Message string into Bytes Array.
8. Define Socket Number and IP address of the Server.
9. Transmit the Request Message to the Server.
10. Start timeout timer, wait for Response. Resend request after timeout has elapsed.

## Server Receive and Response

1. Server is waiting for the client Request message.
2. Decode the Request message. Identify Request ID and Measurement ID.
3. Checks the Request packet format with “Sanity Check” method in DecodeClientMessage class.
4. Integrity check of received message using checksum.
   1. Match Measurement ID from the Measurement array (of DataLookup class).
   2. Find the Index in case there is a match. If no match then Temperature value remains “NaN”.
   3. Goto Temperature array. Find out Temperature value
5. Find appropriate Response Code based on output from step 3, 4 and 5.
6. Prepare Response message string with Request ID, Response Code, Measurement ID, Temperature value.
7. Calculate checksum on Response message string.
8. Append the Response message string with checksum.

Send Response Message.

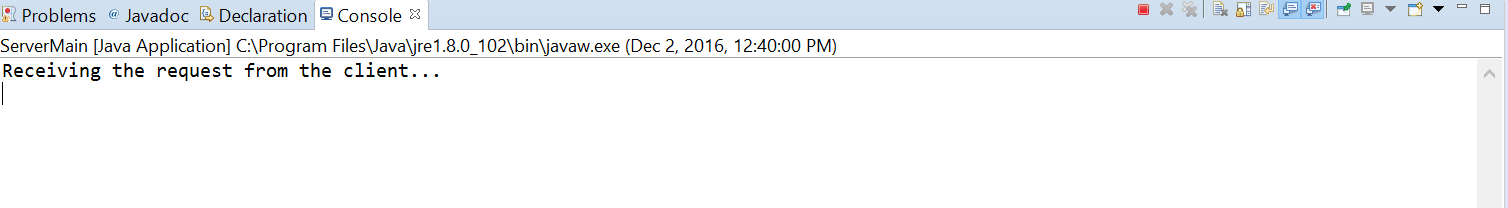
## Client Receive

1. Client is waiting for the server Response message.
2. Decode the Response message. Identify Request ID and Measurement ID, Response ID, Temperature and checksum.
3. Integrity check of received message using checksum.
4. Display the output
   1. 0 The response has been created according to the request.
   2. 1 Error: integrity check failure. The request has one or more bit errors.
   3. 2 Error: malformed request. The syntax of the request message is not correct.
   4. 3 Error: non-existent measurement. The measurement with the requested measurement ID does not exist.
5. Go back to Client Request block.

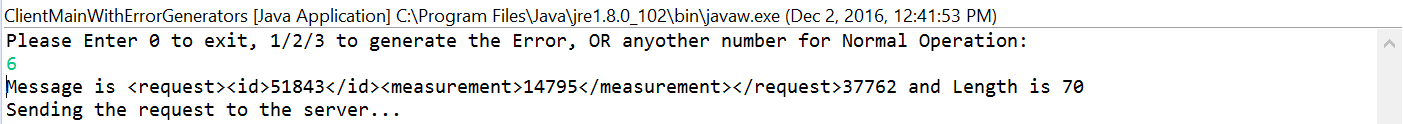
# Test Cases and output

## General output without error

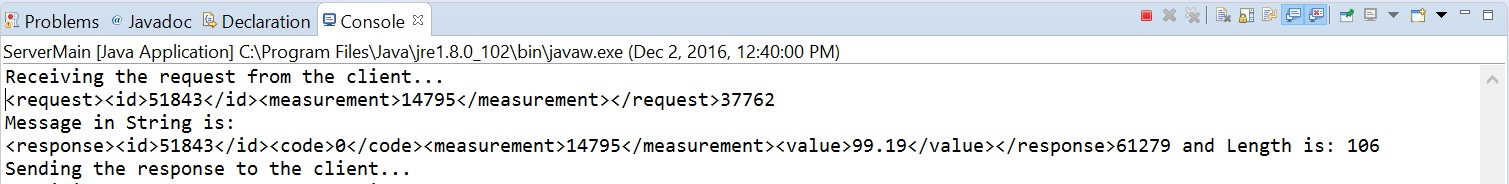
**Step 1** - Server waiting for Request



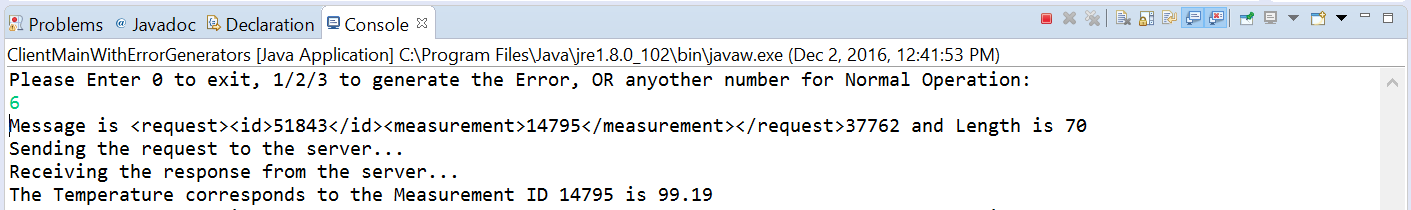
**Step 2** - Client Request



**Step 3** - Server Response

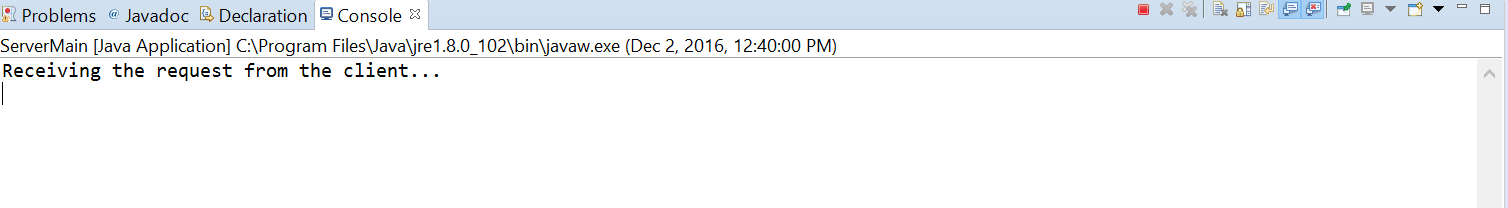


**Step 4** - Client Output

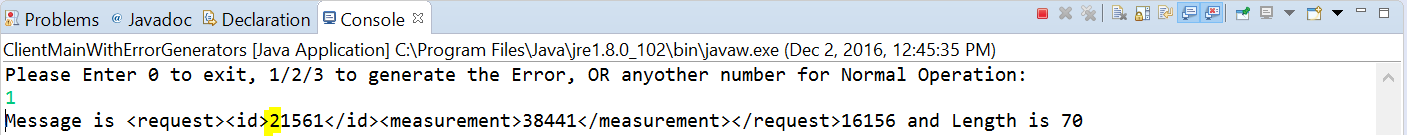


## Integrity Check

**Step 1** - Server waiting for Request

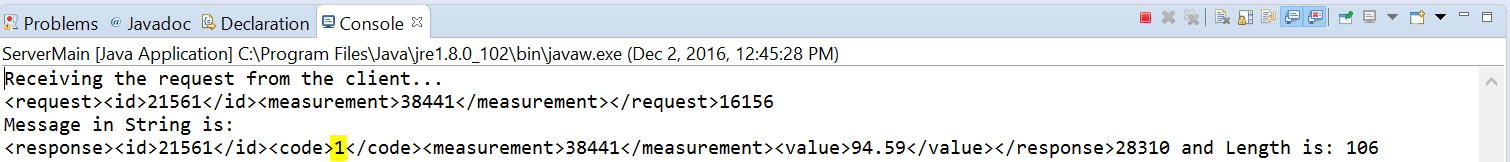


**Step 2** - Client Request



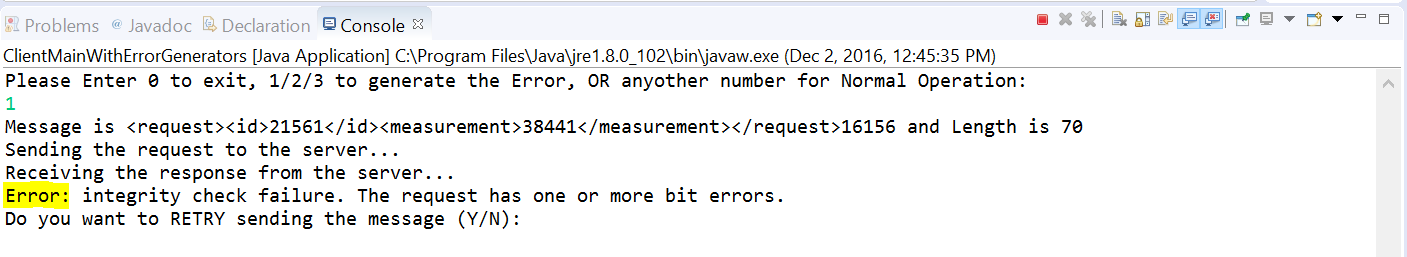
To generate the error, first digit of Request ID (highlighted) is incremented by one if that digit in the Randomly generated number is less than 9. If the digit is 9, then it is set to 1. This makes the checksum invalid. Hence the Server should send Response Code 1.

**Step 3** - Server Response



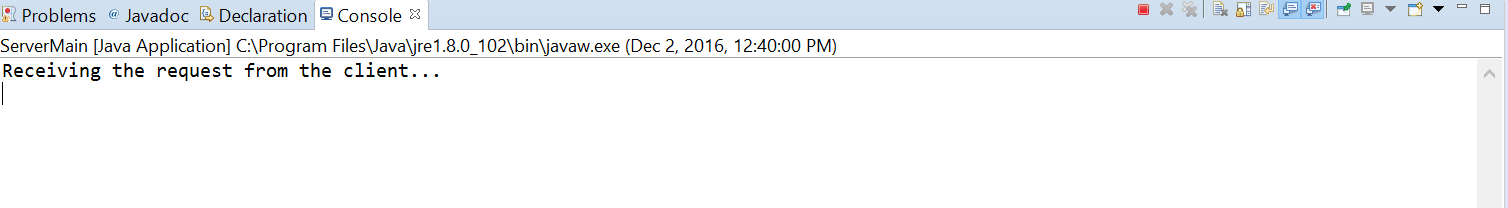
Refer highlighted Response Code 1, which means there was an Integrity Check failure with the Request.

**Step 4** - Client Output

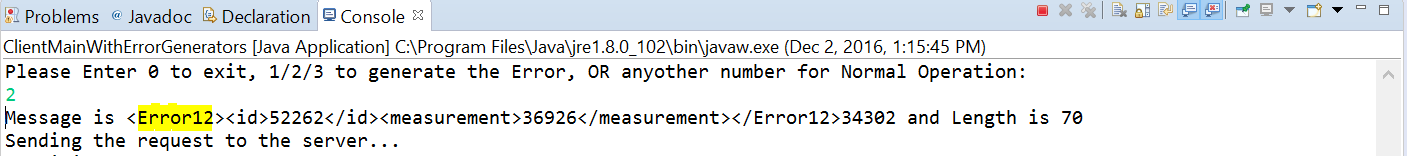


## Malformed Request

**Step 1** - Server waiting for Request

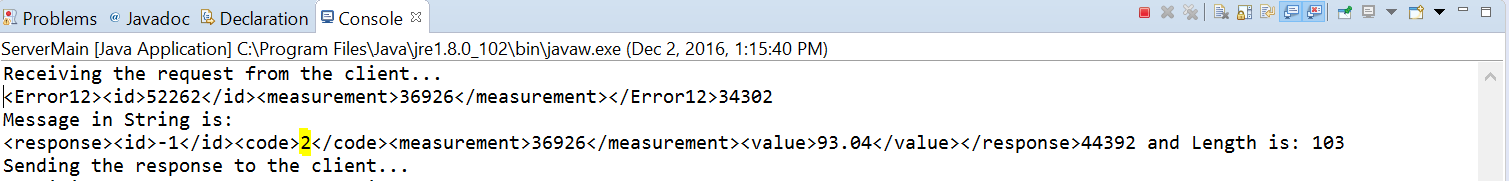


**Step 2** - Client Request



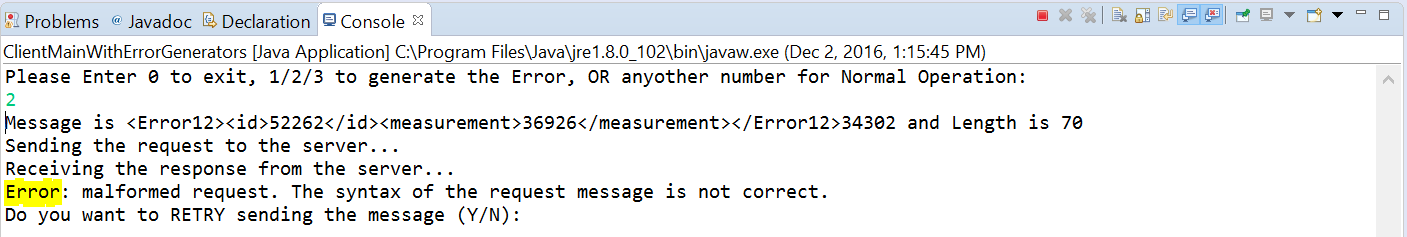
To generate the error, instead of “Request” (highlighted) the client sends “Error12” keyword. This makes the message format invalid. Hence the Server should send Response Code 2.

**Step 3** - Server Response



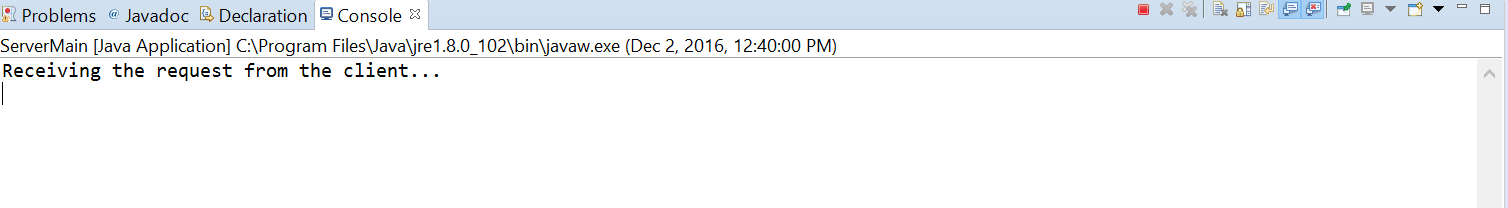
Refer highlighted Response Code 2, which means the Request format was not correct..

**Step 4** - Client Output

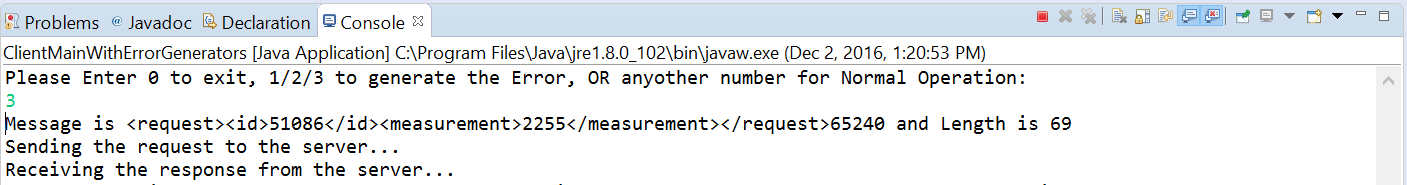


## Non-existant Measurement

**Step 1** - Server waiting for Request

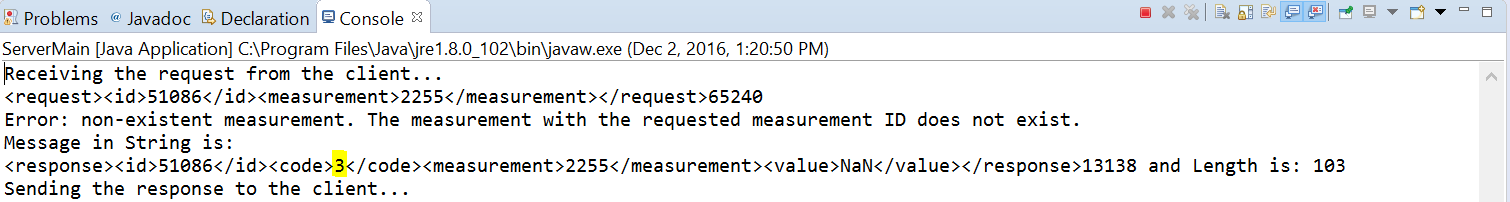


**Step 2** - Client Request



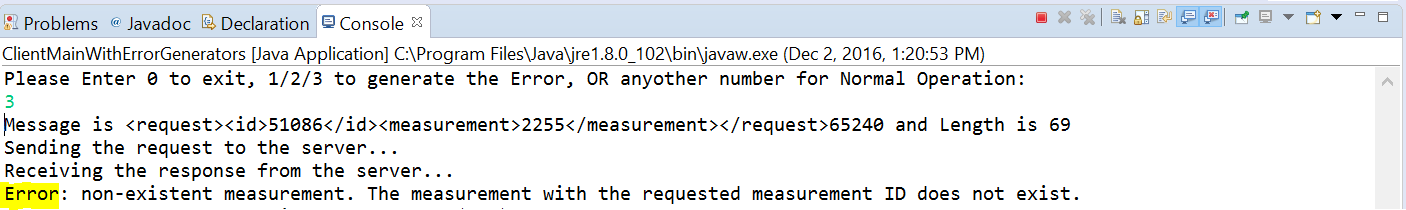
The Client is manually configured to Request Measurement ID “2255” which is not available in the “data.txt” file.

**Step 3** - Server Response



When Server receives request for Measurement ID “2255” and is not able to find a match in Measurement ID array, it generates Response Code 3 and send the Response.

**Step 4** - Client Output

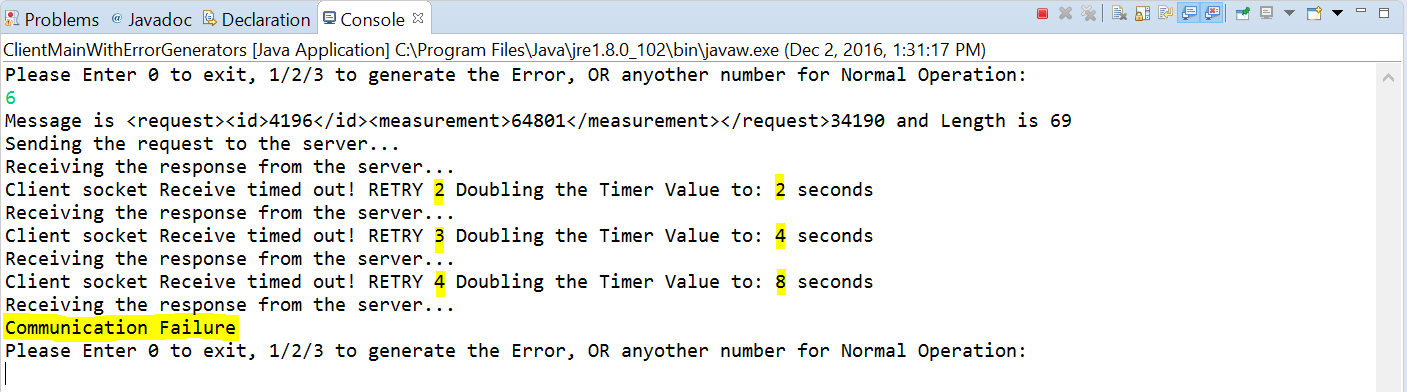


On Receiving the Response ID 3, the Client displays this message.

## Request Timeout

**Step 1** - Server Idle - The program is not running.

**Step 2 -** Client Retry



When Client transmit the Request message it starts the TimeOut timer with default value of 1000. The Clients tries to transmit the Request for maximum of 3 more times (in addition to first one). For each timer expiration, it doubles the Timer. After reaching maximum try, it throws “Communication Failure”.