Data Communications and Security Assignment 2: UDP File Server

Antonio Marotta 101099261

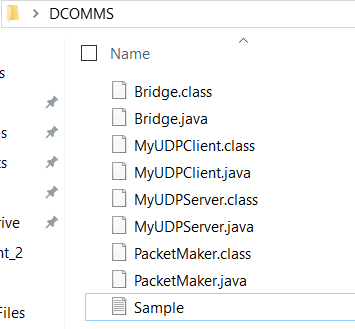
# Level of Complexity

In this assignment I focused on the reliable transfer of data from a server through unreliable channel to a client. The unreliable channel was simulated using a Bridge that was coded to simulate different rdt protocol levels.

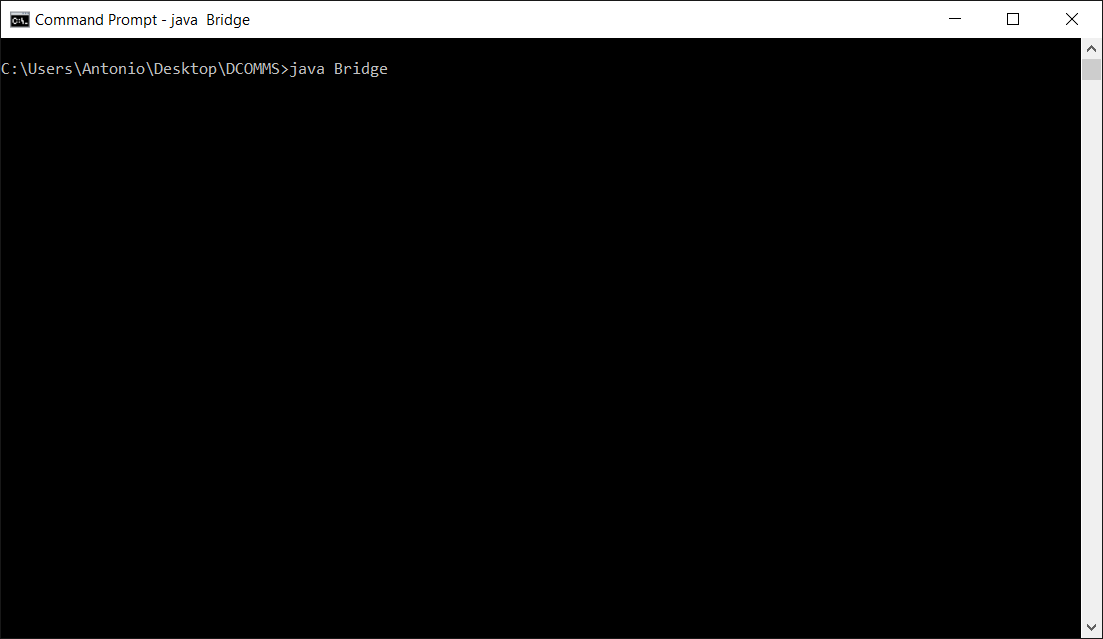
This assignment was developed incrementally, in line with the different rdt levels, where I reached rdt3.0. Therefore, my code can work with the most problematic bridge where both variables = false and there is error, packet loss and ACK loss.

# How to use

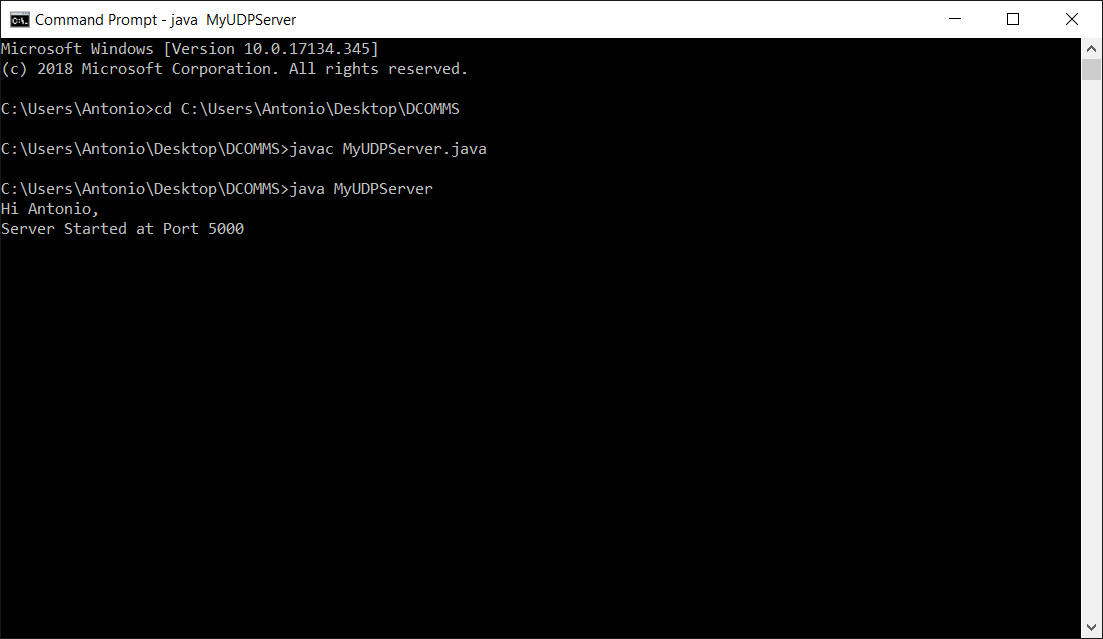
There are four classes that are needed: Bridge.class, MyUDPServer.class, MyUDPClient.class and PacketMaker.class. Make sure the file you want to send is in the same directory as the Bridge e.g. Sample.txt



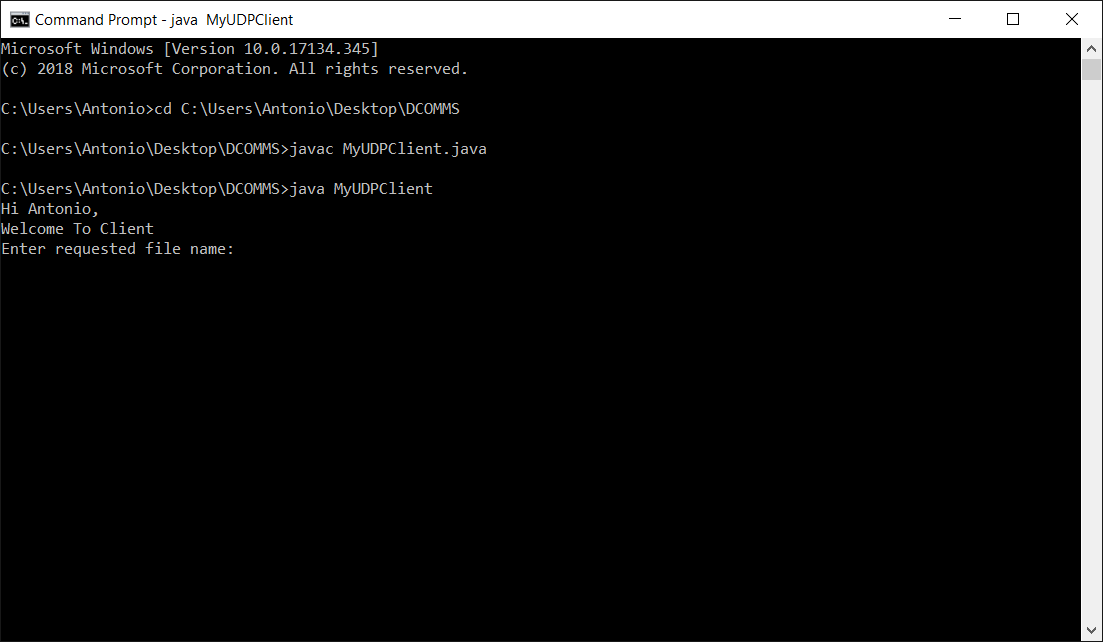
1. Run the Bridge



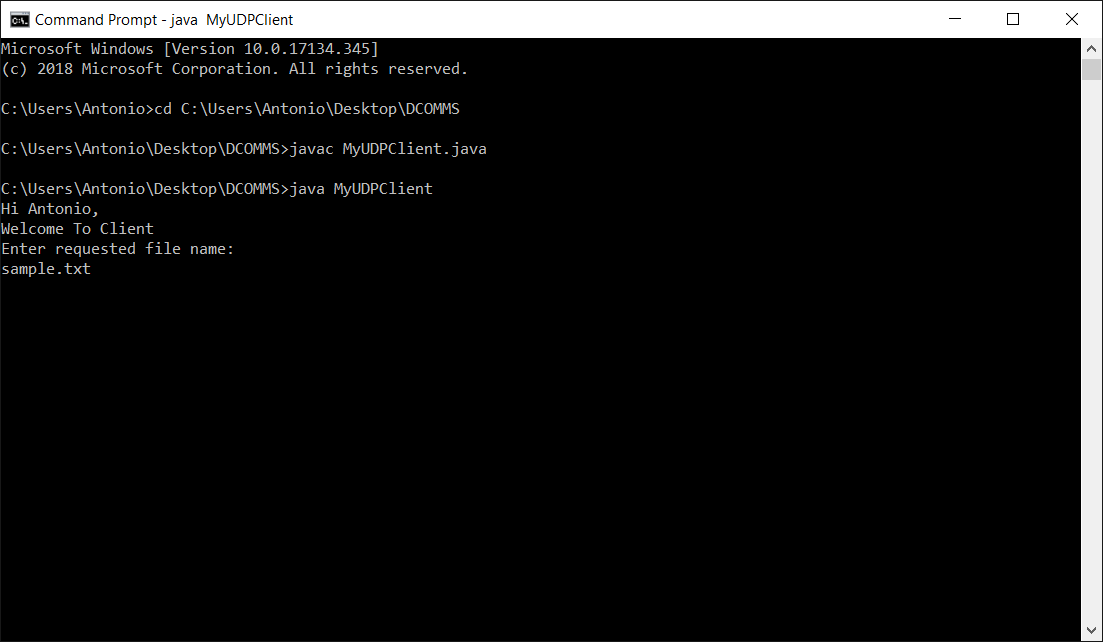
1. Run the Server



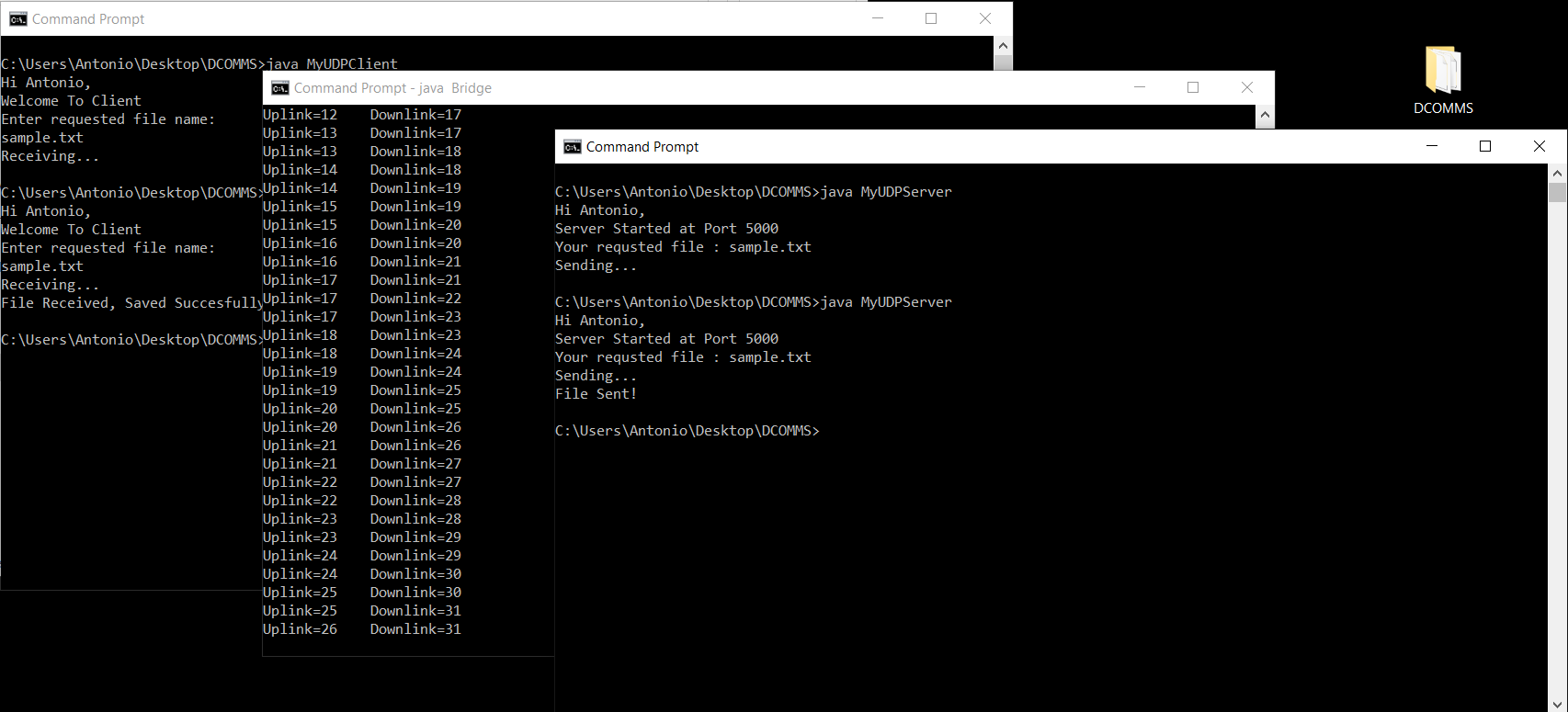
1. Run the Client



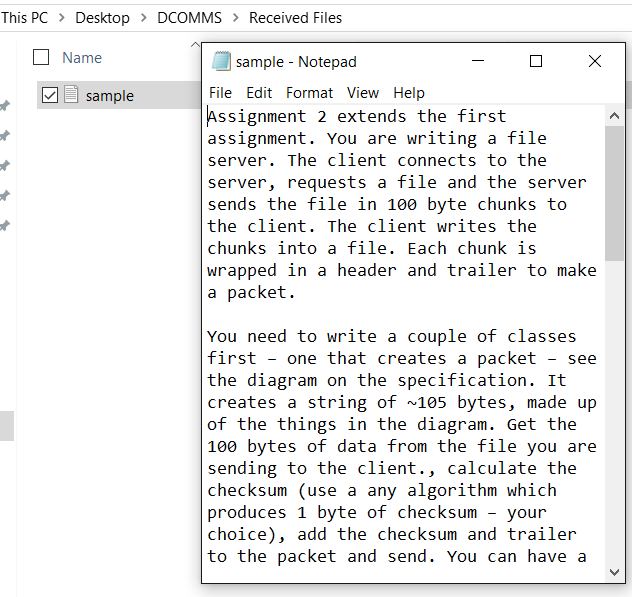
1. Enter the full name of the file you want to receive from the Server e.g. sample.txt



1. Wait for the transfer to complete, the bridge simulates an unreliable channel, so it may take some time as packets have to be resent



1. A folder named “Received Files” will be created in the same directory as the Client. This is where the transferred file will appear.



Note: On rare occasion the text file doesn’t transfer perfectly, this can be fixed with a stronger checksum.

# Code

## Bridge

1. // A stateless bridge between a client and a server simulating UDP packet loss and error.
2. // Author: Dr. Mohammad Rezaeian
3. **import** java.net.\*;
4. **import** java.math.\*;
5. **class** Bridge
6. {
7. **public** **static** **void** main(String str[]) **throws** Exception
8. {
9. **boolean** perfectNetwork=**false**;
10. **boolean** noLoss=**false**; //Packet loss is not possible (only error)
11. **boolean** noAckLoss=**false**;//Packets from client to server impossible to be lost
12. **boolean** s2c=**false**;
13. **int** N=4;//statisticaly a loss or error event happens in every N packets
14. **byte**[] dataBuffer = **new** **byte**[128];
15. **int** bridgePort=4000;
16. **int** serverPort=5000;
17. InetAddress IPAddress = InetAddress.getByName("localhost");
18. DatagramPacket sendPacket = **new** DatagramPacket(dataBuffer, dataBuffer.length, IPAddress, serverPort);
19. DatagramPacket receivePacket = **new** DatagramPacket(dataBuffer, dataBuffer.length);
20. **do**{**try**{ //for avoiding rare binding exceptions
21. DatagramSocket routerSocket = **new** DatagramSocket(bridgePort);
22. routerSocket.receive(receivePacket);
23. **int** clientPort = receivePacket.getPort();
24. **int** downlink=0,uplink=1;
25. **boolean** lost=**false**;
26. **while**(**true**)
27. {
28. **if**(!lost||noLoss)routerSocket.send(sendPacket);
29. lost=**false**;
30. dataBuffer = **new** **byte**[128];
31. receivePacket.setData(dataBuffer);
32. sendPacket.setData(dataBuffer);
33. System.out.println("Uplink="+uplink+"    Downlink="+downlink);
34. routerSocket.receive(receivePacket);
35. **if** (receivePacket.getPort()== clientPort)
36. {sendPacket.setPort(serverPort);uplink++;s2c=**false**;}
37. **else** {sendPacket.setPort(clientPort);downlink++;s2c=**true**;}
38. **if**(!perfectNetwork){
39. **if**(!noAckLoss||(noAckLoss && s2c))
40. **if**( (**int**)(Math.random()\*N) == 1 )lost=**true**;
41. **if**( (**int**)(Math.random()\*N) == 1 )dataBuffer[2]='\*';
42. }
43. }
44. }**catch**(SocketException e){bridgePort++;System.out.println("Note to client: Bridge port is "+bridgePort);}
45. }**while**(**true**);
46. }
47. }

## MyUDPClient

1. /\*
2. A UDP Client with error handling that receives packets from the server, sends an ACK and decodes into a directory.
3. Author: Antonio  Marotta
4. @1.3
5. \*/
6. **import** java.net.\*;
7. **import** java.io.\*;
9. **class** MyUDPClient{
10. **public** **static** **void** main(String str[]){
11. **try**{
12. System.out.println("Hi Antonio,");
13. System.out.println("Welcome To Client");
14. BufferedReader inFromUser =
15. **new** BufferedReader(**new** InputStreamReader(System.in));
16. DatagramSocket clientSocket = **new** DatagramSocket();
17. InetAddress IPAddress = InetAddress.getByName("localhost");
18. **byte**[] sendRequest;
19. String fileRequested;
21. sendRequest = **new** **byte**[1024];
22. System.out.println("Enter requested file name: ");
23. fileRequested = inFromUser.readLine();
24. sendRequest = fileRequested.getBytes();
25. DatagramPacket sendReqPacket = **new** DatagramPacket(sendRequest, sendRequest.length, IPAddress, 4000);
26. clientSocket.send(sendReqPacket);
27. System.out.println("Receiving...");
29. **int** counter = 0;
30. **new** File("Received Files").mkdir();
31. BufferedOutputStream writer = **new** BufferedOutputStream(**new** FileOutputStream("Received Files\\" + fileRequested));
32. **while**(**true**)
33. {
34. **byte**[] receiveData = **new** **byte**[101];
35. DatagramPacket receivePacket = **new** DatagramPacket(receiveData, receiveData.length);
36. clientSocket.receive(receivePacket);
38. //Sending ACK NACK
39. //Gathering it again
40. InetAddress IP = receivePacket.getAddress();
41. **int** port2 = receivePacket.getPort();
42. String ackNack = "ACK";
43. DatagramPacket response = **new** DatagramPacket(ackNack.getBytes(), ackNack.getBytes().length, IP, port2);
44. clientSocket.send(response);
46. String decode = **new** String(receiveData);
48. **if**(decode.contains("Sorry")){
49. System.out.println(decode.trim());
50. **break**;
51. }
53. //Received File
55. **if**(!decode.contains("/q")){
56. **byte**[] toWrite = PacketMaker.removeChecksum(receiveData);
58. //System.out.println(++counter + ") Receiving packet contains 1st byte : " + toWrite[0]);
60. writer.write(toWrite, 0, toWrite.length);
61. writer.flush();
62. }**else**{
63. System.out.println("File Received, Saved Succesfully!");
64. **break**;
65. }
67. }
69. writer.close();
70. clientSocket.close();
71. }**catch**(Exception ex){
72. ex.printStackTrace();
73. }
74. }
75. }

## MyUDPServer

1. /\*
2. A UDP Server with error handling that can send files to a client via small packets.
3. Author: Antonio  Marotta
4. @1.3
5. \*/
6. **import** java.net.\*;
7. **import** java.io.\*;
8. **import** java.util.ArrayList;
10. **class** MyUDPServer{
11. **public** **static** **void** main(String str[]){
12. **try**{
13. DatagramSocket serverSocket = **new** DatagramSocket(**null**);
14. serverSocket.setReuseAddress(**true**);
15. serverSocket.bind(**new** InetSocketAddress(5000));
16. System.out.println("Hi Antonio,");
17. System.out.println("Server Started at Port 5000");
18. **byte**[] receiveData = **new** **byte**[1024];
20. **int** counter = 0;
21. **while**(**true**){
22. //Listening to receive Packet
23. receiveData = **new** **byte**[1024];
24. //System.out.println("Waiting for client's call");
25. DatagramPacket receivePacket = **new** DatagramPacket(receiveData, receiveData.length);
26. serverSocket.receive(receivePacket);
27. String sentence = **new** String( receivePacket.getData()).trim();
29. **if**(!sentence.startsWith("AC") && !sentence.startsWith("NAC")){
31. //Gathering client information
32. InetAddress IPAddress = receivePacket.getAddress();
33. **int** port = receivePacket.getPort();
35. //Packet Received
36. System.out.println("Your requsted file : " + sentence);
38. **if**(**new** File(sentence).exists()){
39. System.out.println("Sending...");
40. ArrayList<**byte**[]> packets = PacketMaker.makePackets(sentence);
41. **for**(**int** i=0; i<packets.size(); i++){
42. **byte**[] sendablePack = PacketMaker.makePacketForSend(packets.get(i));
44. DatagramPacket toBeSend = **new** DatagramPacket(sendablePack, sendablePack.length, IPAddress, port);
45. //System.out.println(++counter + ") Sending packet contains 1st byte : " + packets.get(i)[0]);
46. serverSocket.send(toBeSend);
48. //ACK NACK Handling
49. **byte**[] res = **new** **byte**[3];
50. DatagramPacket response = **new** DatagramPacket(res, res.length);
51. serverSocket.setSoTimeout(2000);
52. **try**{
53. serverSocket.receive(response);
54. }**catch**(SocketTimeoutException e){i--;}
56. }
58. String quitMsg = "/quit";
59. **byte**[] quitMsgBytes = quitMsg.getBytes();
60. DatagramPacket quitIndicator = **new** DatagramPacket(quitMsgBytes, quitMsgBytes.length, IPAddress, port);
61. serverSocket.send(quitIndicator);
62. System.out.println("File Sent!");
63. }**else**{
64. String msg = "Sorry! Server does not contain this file";
65. **byte**[] msgInBytes = msg.getBytes();
66. DatagramPacket noFile = **new** DatagramPacket(msgInBytes, msgInBytes.length, IPAddress, port);
67. serverSocket.send(noFile);
68. }
69. }
70. }
71. }**catch**(Exception ex){
72. //ex.printStackTrace();
73. }
74. }
75. }

## PacketMaker

1. /\*
2. A simple UDP Packet Maker that can make packets and add, check and remove checksum.
3. Author: Antonio  Marotta
4. @1.2
5. \*/
6. **import** java.io.\*;
7. **import** java.util.ArrayList;
9. **class** PacketMaker{
10. **public** **static** **final** **byte** checkSum(**byte**[] bytes) {
11. **byte** sum = 0;
12. **for** (**byte** b : bytes) {
13. sum ^= b;
14. }
15. **return** sum;
16. }
18. **public** **static** **byte**[] makePacketForSend(**byte**[] data){
19. **byte**[] newPacket = **new** **byte**[data.length+1];
20. **for**(**int** i=0; i<data.length; i++){
21. newPacket[i] = data[i];
22. }
23. newPacket[newPacket.length-1] = checkSum(data);
24. **return** newPacket;
25. }
27. **public** **static** ArrayList<**byte**[]> makePackets(String fileName){
28. ArrayList<**byte**[]> Packets = **new** ArrayList<**byte**[]>();
29. **try**{
30. BufferedInputStream reader = **new** BufferedInputStream(**new** FileInputStream(fileName));
31. **byte**[] buff = **new** **byte**[100];
32. **int** len;
33. **while**( (len = reader.read(buff)) != -1){
34. Packets.add(buff);
35. buff = **new** **byte**[100];
36. }
38. }**catch**(Exception ex){
39. System.out.println("File not found");
40. **return** **null**;
41. }
42. **return** Packets;
43. }
45. **public** **static** **boolean** isChecksumCorrect(**byte**[] packet){
46. **byte** carriedCheckSum = packet[packet.length-1];
47. **byte**[] extract = **new** **byte**[packet.length-1];
48. **for**(**int** i=0; i<extract.length; i++){
49. extract[i] = packet[i];
50. }
52. //calculate checksum
53. **byte** newCheckSum = checkSum(extract);
55. **if**(newCheckSum == carriedCheckSum)
56. **return** **true**;
57. **else**
58. **return** **false**;
59. }
61. **public** **static** **byte**[] removeChecksum(**byte**[] packet){
62. **byte**[] newPacket = **new** **byte**[packet.length-1];
63. **for**(**int** i=0; i<newPacket.length; i++)
64. newPacket[i] = packet[i];
65. **return** newPacket;
66. }
67. }