



Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή

Δίκτυα Υπολογιστών II

Θεόδωρος Κατζάλης
AEM:9282
katzalis@auth.gr

May 2, 2021

Contents

1	Δομή του προγράμματος	2
2	UserApplication.java	3
3	applications	9
3.1	Echo.java	9
3.2	Media.java	12
3.3	Obd.java	19
3.4	Copter.java	22

1 Δομή του προγράμματος

```
applications
  Copter.java
  Echo.java
  Media.java
  Obd.java
plots
  plot.ipynb
ascii
  audio.txt
  auto.txt
  copter_tcp.txt
  copter.txt
  echo.txt
  https.txt
  image.txt
  obd_tcp.txt
  obd.txt
  temp.txt
  test.txt
  welcome.txt
UserApplication.java
```

- Το αρχείο που βρίσκεται η main είναι το *UserApplication.java*.
- Στον φάκελο *applications*, δημιουργήσαμε ξεχωριστά αρχεία για κάθε εφαρμογή.
- Στον φάκελο *ascii*, βρίσκονται οι έξοδοι του προγράμματος figlet για ascii art λόγους, όπως έχει αναφερθεί και στο report!
- Στον φάκελο *plots*, έχουμε τέλος ένα python αρχείο για να δημιουργήσουμε τα διαγράμματα μας.

2 UserApplication.java

```
1 import applications.*;
2 import java.io.File;
3 import java.io.FileWriter;
4 import java.io.InputStream;
5 import java.io.OutputStream;
6 import java.lang.System;
7 import java.net.*;
8 import java.nio.charset.StandardCharsets;
9 import java.time.LocalDateTime;
10 import java.util.Scanner;
11
12 class UserApplication {
13
14     public static void main(String[] args) {
15
16         printWelcome();
17
18         String[] codes = WebScraping.getCodes();
19         int clientPort = Integer.valueOf(codes[0]);
20         int serverPort = Integer.valueOf(codes[1]);
21
22         String requestCodeEcho = codes[2];
23         String requestCodeImage = codes[3] + "UDP=1024";
24         String requestCodeSound = codes[4];
25         String requestCodeCopter = codes[5];
26         String requestCodeVehicle = codes[6];
27
28         byte[] hostIP = {(byte)155, (byte)207, (byte)18, (byte)208};
29         InetAddress hostAddress = null;
30         DatagramSocket socket = null;
31
32         try {
33             hostAddress = InetAddress.getByAddress(hostIP);
34             socket = new DatagramSocket(clientPort);
35         } catch (Exception x) {
36             x.printStackTrace();
37         }
38
39         Scanner in = new Scanner(System.in);
40
41         // control user input
42         int flag = 1;
43         do {
44             System.out.println(
45                 "\nPlease enter a number (1-11). Available options are:\n"
46                 + "1) Echo with delay\n"
47                 + "2) Echo no delay\n"
48                 + "3) Temperature\n"
49                 + "4) Image\n"
50                 + "5) Music\n"
51                 + "6) Vehicle UDP\n"
52                 + "7) Ithakicopter UDP\n"
53                 + "8) Autopilot\n"
54                 + "9) HTTPS TCP\n"
55                 + "10) Ithakicopter TCP\n"
56                 + "11) Vehicle TCP");
57             String choiceApp = in.nextLine();
58
```

```

59 switch (choiceApp) {
60 case "1":
61
62     /* ----- Echo with delay ----- */
63     printASCII("src/ascii/echo.txt");
64
65     try (FileWriter writerInfo =
66         new FileWriter(new File("logs/echo_info_delay.txt"))) {
67         writerInfo.write("Info:\n"
68             + "The request code is " + requestCodeEcho + "\n");
69         writerInfo.write("Tic: " + LocalDateTime.now() + "\n");
70
71         Echo.telemetry(socket, hostAddress, serverPort, requestCodeEcho,
72             "delay.txt");
73
74         writerInfo.write("Toc: " + LocalDateTime.now());
75     } catch (Exception x) {
76         x.printStackTrace();
77     }
78
79     break;
80
81 case "2":
82     /* ----- Echo no delay ----- */
83     printASCII("src/ascii/echo.txt");
84
85     try (FileWriter writerInfo =
86         new FileWriter(new File("logs/echo_info_no_delay.txt"))) {
87
88         writerInfo.write("Info:\n"
89             + "The request code is "
90             + "requestCodeEcho"
91             + "\n");
92         writerInfo.write("Tic: " + LocalDateTime.now() + "\n");
93
94         Echo.telemetry(socket, hostAddress, serverPort, "E0000",
95             "no_delay.txt");
96         writerInfo.write("Toc: " + LocalDateTime.now());
97     } catch (Exception x) {
98         x.printStackTrace();
99     }
100
101     break;
102
103 case "3":
104     /* ----- Temperature ----- */
105     printASCII("src/ascii/temp.txt");
106
107     try (FileWriter writerTemp =
108         new FileWriter(new File("logs/temp_info.txt"))) {
109         writerTemp.write("Info Temperature app:\n" + requestCodeEcho + "\n" +
110             LocalDateTime.now() + "\n");
111
112         for (int i = 0; i < 1; i++) {
113             Echo.execute(socket, hostAddress, serverPort,
114                 requestCodeEcho + "T00");
115             System.out.println();
116         }
117
118         writerTemp.write(LocalDateTime.now() + "\n");

```

```

119     } catch (Exception x) {
120         x.printStackTrace();
121     }
122     break;
123
124 case "4":
125     /* ----- Image ----- */
126     printASCII("src/ascii/image.txt");
127
128     String encodingImage = "";
129     System.out.println("Enter 0 for CAM and 1 for PTZ: ");
130     try {
131         int userInput = in.nextInt();
132         if (Integer.valueOf(userInput) == 1) {
133             encodingImage = "CAM=PTZ";
134         } else {
135             encodingImage = "CAM=FIX";
136         }
137     } catch (Exception x) {
138         x.printStackTrace();
139     }
140
141     try (FileWriter writerImage = new FileWriter(
142         new File("logs/image_info_" + encodingImage + ".txt"))) {
143         writerImage.write(encodingImage + "\n" + requestCodeImage + "\n" +
144             LocalDateTime.now() + "\n");
145
146         for (int i = 0; i < 1; i++) {
147             Media.image(socket, hostAddress, serverPort,
148                 requestCodeImage + encodingImage);
149             System.out.println();
150         }
151
152         writerImage.write(LocalDateTime.now() + "\n");
153     } catch (Exception x) {
154         x.printStackTrace();
155     }
156
157     break;
158
159 case "5":
160     /* ----- Audio ----- */
161     printASCII("src/ascii/audio.txt");
162
163     String numAudioPackets = "999";
164
165     // song
166     String type = "F";
167
168     // tone
169     // String type = "T";
170
171     // AQDPCM modulation
172     // String encoding = "AQ";
173
174     // DPCM modulation
175     String encoding = "";
176
177     // choose song L00 - L??
178     String songID = "L02";

```

```

179
180 File infoMusic =
181     new File("logs/music_info_" + encoding + type + ".txt");
182 try (FileWriter writerInfoMusic = new FileWriter(infoMusic)) {
183     writerInfoMusic.write(requestCodeSound + "\nEncoding: " + encoding +
184         "\nType: " + type + LocalDateTime.now() + "\n");
185
186     Media.audio(socket, hostAddress, serverPort, encoding, type,
187         numAudioPackets, songID, requestCodeSound);
188     System.out.println();
189
190     writerInfoMusic.write(LocalDateTime.now() + "\n");
191     writerInfoMusic.close();
192 } catch (Exception x) {
193     x.printStackTrace();
194 }
195
196 break;
197
198 case "6":
199     /* ----- Vehicle OBD UDP ----- */
200     printASCII("src/ascii/obd.txt");
201     Obd.udpTelemetry(socket, hostAddress, serverPort, requestCodeVehicle);
202     break;
203
204 case "7":
205     /* ----- Ithakicopter UDP ----- */
206     printASCII("src/ascii/copter.txt");
207
208     try {
209         socket = new DatagramSocket(48078);
210     } catch (Exception x) {
211         x.printStackTrace();
212     }
213
214     copterWelcome();
215
216     try (FileWriter writerCopter =
217         new FileWriter(new File("logs/copter_info.txt"))) {
218         writerCopter.write("Info Ithakicopter app:\n" + LocalDateTime.now() +
219             "\n");
220         writerCopter.write("MOTOR ALTITUDE TEMPERATURE PRESSURE");
221
222         while (System.in.available() == 0) {
223             Copter.udpTelemetry(socket, hostAddress, serverPort, writerCopter);
224         }
225
226         writerCopter.write(LocalDateTime.now() + "\n");
227     } catch (Exception x) {
228         x.printStackTrace();
229     }
230     break;
231
232 case "8":
233     /* ----- Autopilot ----- */
234     printASCII("src/ascii/auto.txt");
235
236     try (Socket socketAuto = new Socket(hostAddress, 38048)) {
237         socket = new DatagramSocket(48078);
238         int lowerBound = 160;

```

```

239     int higherBound = 190;
240     Copter.autopilot(socket, hostAddress, serverPort, socketAuto,
241                     Math.min(200, Math.max(150, lowerBound)),
242                     Math.min(200, Math.max(150, higherBound)));
243 } catch (Exception x) {
244     x.printStackTrace();
245     ;
246 }
247
248 break;
249
250 case "9":
251     /* ----- HTTPS TCP ----- */
252     printASCII("src/ascii/https.txt");
253
254     try (Socket httpsSocket = new Socket(hostAddress, 80)) {
255         https(httpsSocket);
256     } catch (Exception x) {
257         x.printStackTrace();
258     }
259
260     break;
261
262 case "10":
263     /* ----- IthakicopterTCP ----- */
264     printASCII("src/ascii/copter_tcp.txt");
265
266     int target = 180;
267     for (int i = 0; i < 10; i++) {
268         System.out.println(
269             new String(Copter.tcpTelemetry(hostAddress, target)));
270     }
271
272     break;
273
274 case "11":
275     /* ----- Vehicle OBD TCP ----- */
276     printASCII("src/ascii/obd_tcp.txt");
277
278     try (Socket socketVehicle = new Socket(hostAddress, 29078);
279         FileWriter writerVehicleInfo =
280             new FileWriter(new File("logs/car_info.txt"));
281         FileWriter writerVehicleData =
282             new FileWriter(new File("logs/car_telemetry.txt"))) {
283
284         writerVehicleInfo.write("Info Vehicle app:\n" + LocalDateTime.now() +
285                                "\n");
286
287         final int minutes = 2;
288         final int secondsPerMinute = 60;
289         final int timeInterval = minutes * secondsPerMinute;
290         float engineTime = 0;
291         while (engineTime < timeInterval) {
292             engineTime = Obd.tcpTelemetry(socketVehicle, writerVehicleData);
293             System.out.println("The engine run time is " + engineTime + "\n");
294         }
295
296         writerVehicleInfo.write(LocalDateTime.now() + "\n");
297     } catch (Exception x) {
298         x.printStackTrace();

```



```

299     }
300
301     break;
302
303     default:
304         System.out.println(
305             "Please provide a valid input. If you want to exit then press Control-C.\n");
306         flag = 0;
307     }
308 } while (flag == 0);
309
310 /* ----- Close streams ----- */
311 if (socket != null) {
312     try {
313         socket.close();
314         in.close();
315     } catch (Exception x) {
316         x.printStackTrace();
317     }
318 }
319 System.out.println("\nShuting down UDP sockets...");
320
321 System.out.println(
322     "\nx-----Hooray! Java application finished successfully
323     !-----x");
324 }
325
326 /**
327  * Print ASCII text
328  * @param filePath The path of the file with the ASCII characters to be
329  *     printed
330  */
331 private static void printASCII(String filePath) {
332     try {
333         Scanner input = new Scanner(new File(filePath));
334         while (input.hasNextLine()) {
335             System.out.println(input.nextLine());
336         }
337         Thread.sleep(1500); // pause a little bit to enjoy the view
338     } catch (Exception x) {
339         x.printStackTrace();
340     }
341 }
342
343 private static void copterWelcome() {
344     System.out.println(
345         "For Ithakicopter UDP telemetry you need to open ithakicopter.jar");
346     System.out.print("Did you open it? If yes press ENTER to continue");
347     try {
348         System.in.read();
349         Thread.sleep(1000); // pause a bit to catch up with the user
350         System.out.println("Press ENTER to exit");
351         Thread.sleep(1000);
352     } catch (Exception x) {
353         x.printStackTrace();
354     }
355 }
356
357 /**
358  * Print welcome screen ASCII with CYAN color

```

```

358 */
359 private static void printWelcome() {
360     // windows users may be not able to view colors on terminal
361     final String ANSI_CYAN = "\u001B[36m";
362     final String ANSI_RESET = "\u001B[0m";
363
364     try (Scanner input = new Scanner(new File("src/ascii/welcome.txt"))) {
365         while (input.hasNextLine()) {
366             System.out.print(ANSI_CYAN); // add some color!
367             System.out.print(input.nextLine());
368             System.out.println(ANSI_RESET);
369         }
370         System.out.println();
371         System.out.print("Press ENTER to continue");
372         System.in.read(); // pause a little bit to enjoy the view
373
374     } catch (Exception x) {
375         x.printStackTrace();
376     }
377 }
378
379 private static void https(Socket socket) {
380     try {
381         InputStream in =
382             socket.getInputStream(); // what I receive from the server
383         OutputStream out = socket.getOutputStream(); // what i send to the server
384
385         long timeBefore = System.currentTimeMillis();
386         out.write(
387             "GET /netlab/hello.html HTTP/1.0\r\nHost: ithaki.eng.auth.gr:80\r\n\r\n"
388             .getBytes());
389
390         byte[] inputBuffer = in.readAllBytes();
391         String message = new String(inputBuffer, StandardCharsets.US_ASCII);
392         System.out.println("Ithaki responded via TCP with: \n" + message);
393         System.out.println(
394             "Time response: " +
395             (System.currentTimeMillis() - timeBefore) / (float)1000 + " seconds");
396         socket.close();
397     } catch (Exception x) {
398         x.printStackTrace();
399     }
400 }
401 }

```

3 applications

3.1 Echo.java

```

1 package applications;
2
3 import java.io.File;
4 import java.io.FileWriter;
5 import java.net.DatagramPacket;
6 import java.net.DatagramSocket;
7 import java.net.InetAddress;
8 import java.nio.charset.StandardCharsets;
9
10 public class Echo {
11

```

```

12  /*
13  * UDP TX/RX Echo application with delay
14  *
15  * WARNING: It doesn't close the DatagramSocket. You should do it manually if
16  * it is desired after the call of the function.
17  *
18  * @param requestCode If request code is set to E000 then the execute will
19  *     have no delay for the RX
20  */
21  public static long execute(DatagramSocket socket, InetAddress hostAddress,
22                           int serverPort, String requestCode) {
23      System.out.println(
24          "\n-----Echo application-----");
25
26      if (requestCode.equals("E000"))
27          System.out.println("Delay: OFF");
28      else if (requestCode.length() > 5)
29          System.out.println("Mode: Temperature\nDelay: OFF");
30      else
31          System.out.println("Delay: ON");
32
33      byte[] txbuffer = requestCode.getBytes();
34      byte[] rxbuffer = new byte[64];
35      long diff = 0;
36      try {
37          socket.setSoTimeout(3000);
38          DatagramPacket sendPacket = new DatagramPacket(txbuffer, txbuffer.length,
39                                                         hostAddress, serverPort);
40          DatagramPacket receivePacket =
41              new DatagramPacket(rxbuffer, rxbuffer.length);
42
43          // ACTION
44          socket.send(sendPacket);
45          System.out.println(
46              "The request code is: " + requestCode +
47              "\nThe destination port is: " + serverPort +
48              "\nMy listening port (clientPort): " + socket.getLocalPort());
49          long timeBefore = System.currentTimeMillis();
50
51          // LISTEN
52          socket.receive(receivePacket);
53          long timeAfter = System.currentTimeMillis();
54          diff = timeAfter - timeBefore;
55          System.out.println("The time required to receive a packet is: " + diff +
56                             " milliseconds");
57          String message =
58              new String(receivePacket.getData(), StandardCharsets.US_ASCII);
59          System.out.println("Ithaki responded with: " + message);
60      } catch (Exception x) {
61          x.printStackTrace();
62      }
63      return diff;
64  }
65
66  /**
67   * For a specific time interval send and receive multiple echo packets
68   * and 1) calculate throughput with moving average (8 seconds window)
69   * and 2) calculate Retransmission Timeout
70   *
71   */

```

```

72 public static void telemetry(DatagramSocket socket, InetAddress hostAddress,
73                             int serverPort, String requestCode,
74                             String fileSuffix) {
75
76     FileWriter fileRto = null;
77     try (FileWriter fileSamples =
78         new FileWriter(new File("logs/echo_samples_" + fileSuffix));
79         FileWriter fileThroughput =
80             new FileWriter(new File("logs/echo_throughput_" + fileSuffix))) {
81
82         if (!requestCode.equals("E0000"))
83             fileRto = new FileWriter(new File("logs/rto.txt"));
84
85         // keep track how many 8 seconds have passed
86         int count8sec[] = new int[8];
87
88         float throughput[] = new float[8];
89
90         int cumulativeSum[] = new int[8];
91
92         long tic[] = new long[8];
93         long timeBefore = System.currentTimeMillis();
94         for (int i = 0; i < 8; i++) {
95             tic[i] = timeBefore + i * 1000; // move per second
96         }
97
98         int isFirst = 1;
99         double rtts = 0;
100        double rttd = 0;
101        double rto = 0;
102
103        final int minutes = 2;
104        final int seconds = 60;
105        int timeInterval = seconds * 1000 * minutes;
106        while ((System.currentTimeMillis() - timeBefore) < timeInterval) {
107            long rtt = Echo.execute(socket, hostAddress, serverPort, requestCode);
108            fileSamples.write(rtt + "\n");
109
110            // throughput moving average
111            throughputCalc(count8sec, throughput, cumulativeSum, tic, timeBefore,
112                          fileThroughput);
113
114            // Retransmission timeout
115            if (!requestCode.equals("E0000")) {
116                rto(isFirst, rtt, rtts, rttd, rto, fileRto);
117                isFirst = 0;
118            }
119        }
120    } catch (Exception x) {
121        x.printStackTrace();
122    } finally {
123        if (fileRto != null) {
124            try {
125                fileRto.close();
126            } catch (Exception x) {
127                x.printStackTrace();
128            }
129        }
130    }
131 }

```

```

132 private static void rto(int isFirst, double rtt, double rtts, double rtttd,
133                         double rto, FileWriter fileRto) throws Exception {
134     // init values
135     if (isFirst == 1) {
136         rtts = rtt;
137         rtttd = rtt / 2;
138         rto = 1; // according to rfc
139         fileRto.write("RTT SRTT RTTd RTO\n");
140     }
141
142     // TODO: Fix magic numbers
143     double temp = rtts;
144     rtts = 0.875 * temp + 0.125 * rtt;
145
146     temp = rtttd;
147     rtttd = 0.75 * temp + 0.25 * Math.abs(rtt - rtts);
148
149     rto = rtts + 1.8 * rtttd;
150     fileRto.write(rtt + " " + rtts + " " + rtttd + " " + rto + "\n");
151
152     System.out.println();
153 }
154
155 private static void throughputCalc(int[] count8sec, float[] throughput,
156                                   int[] cumulativeSum, long[] tic,
157                                   long timeBefore, FileWriter fileThroughput)
158                                   throws Exception {
159     for (int i = 0; i < 8; i++) {
160         long toc = System.currentTimeMillis() - tic[i];
161         System.out.println("The element " + i + " has toc: " + toc);
162         if (toc < 8000 && toc > 0) {
163             // assume no timeouts during the measurements
164             cumulativeSum[i] += 32 * 8;
165             System.out.println("Cumsum: " + cumulativeSum[i]);
166         } else if (toc > 8000) {
167             count8sec[i]++;
168             tic[i] = count8sec[i] * 8000 + timeBefore + i * 1000;
169             throughput[i] = cumulativeSum[i] / (float)8;
170             fileThroughput.write(throughput[i] + "\n");
171
172             System.out.println("I will flush " + cumulativeSum[i] + " cumsum");
173             System.out.println("The throughput is: " + throughput[i]);
174
175             cumulativeSum[i] = 0; // let's start again for the next 8 seconds
176         }
177     }
178 }
179 }
180 }

```

3.2 Media.java

```

1 package applications;
2
3 import java.awt.Desktop;
4 import java.io.ByteArrayInputStream;
5 import java.io.ByteArrayOutputStream;
6 import java.io.File;
7 import java.io.FileOutputStream;
8 import java.io.FileWriter;
9 import java.net.DatagramPacket;

```

```

10 import java.net.DatagramSocket;
11 import java.net.InetAddress;
12 import javax.sound.sampled.*;
13
14 public class Media {
15
16     private static String pathFileImage = "./media/image/ithaki_image.jpg";
17     private static String pathFileSound = "./media/music/track.wav";
18
19     public static void image(DatagramSocket socket, InetAddress hostAddress,
20                             int serverPort, String requestCode) {
21         byte[] txbufferImage = requestCode.getBytes();
22         byte[] rxbufferImage = new byte[1024];
23         int countPackets = 0;
24         long timeBefore = System.currentTimeMillis();
25
26         System.out.println("The request code is " + requestCode);
27         DatagramPacket sendPacket = new DatagramPacket(
28             txbufferImage, txbufferImage.length, hostAddress, serverPort);
29         DatagramPacket receivePacket =
30             new DatagramPacket(rxbufferImage, rxbufferImage.length);
31
32         // TX
33         try {
34             socket.send(sendPacket);
35             if (requestCode.contains("DIR")) {
36                 System.out.println("I am sleeping... Camera needs time to readjust");
37                 Thread.sleep(5000); // sleep in order for the camera to readjust
38             }
39         } catch (Exception x) {
40             x.printStackTrace();
41         }
42
43         // RX
44         ByteArrayOutputStream bufferImage = new ByteArrayOutputStream();
45     outerloop:
46         try {
47             socket.setSoTimeout(3000);
48             for (;;) {
49                 socket.receive(receivePacket); // blocking command
50                 countPackets++;
51
52                 for (int i = 0; i < rxbufferImage.length; i++) {
53                     // System.out.print(String.format("%02X", rxbufferImage[i]));
54                     bufferImage.write(rxbufferImage[i]); // dynamic byte allocation
55                     if ((String.format("%02X", rxbufferImage[i]).equals("D9")) &&
56                         (i != 0)) {
57                         if ((String.format("%02X", rxbufferImage[i - 1]).equals("FF"))) {
58                             break outerloop; // stop writing when EOF (0xFFD9 delimiter)
59                         }
60                     }
61                 }
62             }
63         } catch (Exception x) {
64             x.printStackTrace();
65         }
66
67         byte[] completeDataImage = bufferImage.toByteArray();
68         imageInfo(completeDataImage, countPackets, timeBefore);
69

```

```

70 // save image to a file
71 saveImage(completeDataImage, pathFileImage);
72
73 // openImage(pathFileImage);
74 }
75
76 public static void audio(DatagramSocket socket, InetAddress hostAddress,
77                          int serverPort, String encoding, String type,
78                          String numAudioPackets, String songID,
79                          String requestCodeSound) {
80
81     String completeRequest =
82         requestCodeSound + encoding + type + numAudioPackets;
83     System.out.println("The request code: " + completeRequest);
84
85     // TX
86     byte[] txbufferSound = (songID + completeRequest).getBytes();
87     DatagramPacket sendPacket = new DatagramPacket(
88         txbufferSound, txbufferSound.length, hostAddress, serverPort);
89     try {
90         socket.send(sendPacket);
91     } catch (Exception x) {
92         x.printStackTrace();
93     }
94
95     // RX
96     byte[] dataSound = new byte[128];
97     DatagramPacket receivePacket =
98         new DatagramPacket(dataSound, dataSound.length);
99     ByteArrayOutputStream bufferSound = new ByteArrayOutputStream();
100     int countPackets = 0;
101     int packetSize = 0;
102     long timeBefore = System.currentTimeMillis();
103
104     try (FileWriter writerDiffSamples = new FileWriter(
105         new File("logs/" + encoding + type + "diff_samples.txt"));
106         FileWriter writerSamples = new FileWriter(
107             new File("logs/" + encoding + type + "samples.txt"));
108         FileWriter writerMean =
109             new FileWriter(new File("logs/aqdpcm_mean.txt"));
110         FileWriter writerStep =
111             new FileWriter(new File("logs/aqdpcm_step.txt"))) {
112         socket.setSoTimeout(3000);
113         for (int l = 0; l < Integer.valueOf(numAudioPackets); l++) {
114             socket.receive(receivePacket);
115             countPackets++;
116             packetSize += dataSound.length;
117
118             if (encoding.equals("")) {
119                 bufferSound.write(dpcm(dataSound, writerDiffSamples, writerSamples));
120             } else if (encoding.equals("AQ")) {
121                 bufferSound.write(adpcm(dataSound, writerDiffSamples, writerSamples,
122                     writerMean, writerStep));
123             } else {
124                 System.out.println("This is not a valid request code");
125             }
126         }
127     } catch (Exception x) {
128         x.printStackTrace();
129     }

```

```

130 long timeAfter = System.currentTimeMillis();
131 byte[] completeDataSound = bufferSound.toByteArray();
132 musicInfo(completeDataSound, timeBefore, timeAfter, countPackets,
133           packetSize);
134
135
136 // only in 16 bit samples does matter. In
137 // AQ-DPCM we use 16 bit encoding
138 boolean isBigEndian = false;
139 int encodingBits = 8;
140 if (encoding.equals("AQ")) {
141     isBigEndian = true;
142     encodingBits = 16;
143 }
144
145 AudioFormat modulationPCM =
146     new AudioFormat(8000, encodingBits, 1, true, isBigEndian);
147 // play sound
148 playMusic(completeDataSound, modulationPCM);
149
150 // save music to file
151 saveMusic(completeDataSound, modulationPCM);
152 }
153
154 private static void openImage(String fileImage) {
155     Desktop desktop = Desktop.getDesktop();
156     File imageFile = new File(pathFileImage);
157     if (imageFile.exists()) {
158         try {
159             desktop.open(imageFile);
160         } catch (Exception x) {
161             x.printStackTrace();
162         }
163     }
164 }
165
166 private static void imageInfo(byte[] completeDataImage, int countPackets,
167                               long timeBefore) {
168     // printImageHex(completeDataImage);
169     System.out.println("\nTotal number of packages: " + (countPackets));
170     System.out.println("How many Kbytes is the image? " +
171                       completeDataImage.length / (float)1000);
172
173     System.out.println("Total amount of time to receive a frame: " +
174                       (System.currentTimeMillis() - timeBefore) / (float)1000 +
175                       " seconds");
176     System.out.println(
177         "Total amount of time to receive and write a frame in a .jpg file: " +
178         (System.currentTimeMillis() - timeBefore) / (float)1000 + " seconds");
179 }
180
181 /**
182  * For deubgging purposes print bytes to hexadecimal
183  * @param completeData The byte array to be printed as hexadecimal
184  */
185 private static void printByteHex(byte[] completeData) {
186     System.out.println(
187         "\nComplete byte content of the data file in hexadecimal format:");
188     for (byte i : completeData) {
189         System.out.print(String.format("%02X", i));

```



```

190     }
191 }
192
193 private static void musicInfo(byte[] completeDataSound, long timeBefore,
194                               long timeAfter, int countPackets,
195                               int packetSize) {
196
197     // printByteHex(completeDataSound);
198
199     System.out.println("\n\nTotal number of packages: " + (countPackets));
200     System.out.println(
201         "How many Kbytes is the sound? " +
202         completeDataSound.length / (float)1000 +
203         "\nHow many Kbytes is the data that was actually sent? " +
204         packetSize / (float)1000);
205     System.out.println("Total amount of time to receive sound data: " +
206         (timeAfter - timeBefore) / (float)1000 + " seconds");
207 }
208
209 private static void saveImage(byte[] completeDataImage,
210                               String pathFileImage) {
211     try (FileOutputStream fos = new FileOutputStream(new File(pathFileImage))) {
212         fos.write(completeDataImage);
213         System.out.println("File has been written successfully");
214     } catch (Exception x) {
215         x.printStackTrace();
216     }
217 }
218
219 private static void saveMusic(byte[] completeDataSound,
220                               AudioFormat modulationPCM) {
221     try (AudioInputStream streamSoundInput = new AudioInputStream(
222         new ByteArrayInputStream(completeDataSound), modulationPCM,
223         completeDataSound.length / modulationPCM.getFrameSize())) {
224         AudioSystem.write(streamSoundInput, AudioFileFormat.Type.WAVE,
225             new File(pathFileSound));
226         System.out.println("Sound file creation success");
227     } catch (Exception x) {
228         x.printStackTrace();
229     }
230 }
231
232 private static void playMusic(byte[] completeDataSound,
233                               AudioFormat modulationPCM) {
234     try (SourceDataLine outputAudio =
235         AudioSystem.getSourceDataLine(modulationPCM)) {
236         // outputAudio.open(modulationPCM, 3200);
237         outputAudio.open(modulationPCM);
238         outputAudio.start();
239
240         System.out.println("Getting ready to hear some music?");
241         Thread.sleep(2000);
242         System.out.print("In 3");
243         Thread.sleep(1000);
244         System.out.print(", 2");
245         Thread.sleep(1000);
246         System.out.println(", 1...");
247         Thread.sleep(500);
248         System.out.println("Listening...");
249         Thread.sleep(500);

```

```

250     outputAudio.write(completeDataSound, 0, completeDataSound.length);
251     outputAudio.stop();
252     System.out.println("\nSound application success!");
253 } catch (Exception x) {
254     x.printStackTrace();
255 }
256 }
257
258 private static byte[] dpcm(byte[] dataSound, FileWriter writerDiffSamples,
259                             FileWriter writerSamples) {
260
261     ByteArrayOutputStream bufferSound = new ByteArrayOutputStream();
262     int init = 0;
263     int step = 1;
264
265     for (int i = 0; i < dataSound.length; i++) {
266         // get nibbles
267         int maskLow = 0x0F;
268         int maskHigh = 0xF0;
269
270         // D[i] = x[i] - x[i-1]
271         int nibbleLow = dataSound[i] & maskLow;
272
273         // D[i-1] = x[i-1] - x[i-2]
274         int nibbleHigh = (dataSound[i] & maskHigh) >> 4;
275
276         // differences
277         int diffHigh = (nibbleHigh - 8) * step;
278         int diffLow = (nibbleLow - 8) * step;
279
280         // get samples
281         int sampleFirst = init + diffHigh;
282         int sampleSecond = sampleFirst + diffLow;
283         init = sampleSecond;
284
285         // clipping
286         int[] samples = {sampleFirst, sampleSecond};
287         clipping(samples);
288
289         // write to buffer
290         byte[] decodedSound = new byte[2];
291         decodedSound[0] = (byte)sampleFirst;
292         decodedSound[1] = (byte)sampleSecond;
293
294         try {
295             bufferSound.write(decodedSound);
296             writerDiffSamples.write(diffHigh + "\n" + diffLow + "\n");
297             writerSamples.write(samples[0] + "\n" + samples[1] + "\n");
298         } catch (Exception x) {
299             x.printStackTrace();
300         }
301     }
302
303     try {
304         bufferSound.close();
305     } catch (Exception x) {
306         x.printStackTrace();
307     }
308
309     return bufferSound.toByteArray();

```

```

310 }
311
312 private static byte[] adpcm(byte[] dataSound, FileWriter writerDiffSamples,
313                             FileWriter writerSamples, FileWriter writerMean,
314                             FileWriter writerStep) {
315
316     // get the header first
317     int meanSigned = (dataSound[1] << 8 | dataSound[0]);
318     int step = (Byte.toUnsignedInt(dataSound[3]) << 8 |
319                Byte.toUnsignedInt(dataSound[2]));
320     System.out.println(meanSigned);
321
322     try {
323         writerMean.write(meanSigned + "\n");
324         writerStep.write(step + "\n");
325     } catch (Exception x) {
326         x.printStackTrace();
327     }
328
329     ByteArrayOutputStream bufferSound = new ByteArrayOutputStream();
330     // in DPCM we don't know the init value, we assume
331     // zero. But here we have data in the header.
332     int init = meanSigned;
333     for (int i = 3; i < dataSound.length; i++) {
334         // get nibbles
335         int maskLow = 0x0F;
336         int maskHigh = 0xF0;
337
338         // D[i] = x[i] - x[i-1], should be unsigned
339         int nibbleLow = (dataSound[i] & maskLow);
340
341         // D[i-1] = x[i-1] - x[i-2], should be unsigned
342         int nibbleHigh = (dataSound[i] & maskHigh) >> 4;
343
344         // differences
345         int diffHigh = (nibbleHigh - 8) * step;
346         int diffLow = (nibbleLow - 8) * step;
347
348         // get samples (implement recursive formula)
349         int sampleFirst = init + diffHigh;
350         int sampleSecond = sampleFirst + diffLow;
351         init = sampleSecond;
352
353         // clipping
354         int[] samples = {sampleFirst, sampleSecond};
355         clipping(samples);
356
357         // write data to files
358         try {
359             writerDiffSamples.write(diffHigh + "\n" + diffLow + "\n");
360             writerSamples.write(samples[0] + "\n" + samples[1] + "\n");
361         } catch (Exception x) {
362             x.printStackTrace();
363         }
364
365         // write to buffer
366         byte[] decodedSound = new byte[4];
367         decodedSound[0] = (byte)(samples[0] >> 8); // MSB of sample 15-8
368         decodedSound[1] = (byte)samples[0]; // LSB of sample 7-0
369         decodedSound[2] = (byte)(samples[1] >> 8);

```

```

370     decodedSound[3] = (byte)samples[1];
371
372     try {
373         bufferSound.write(decodedSound);
374     } catch (Exception x) {
375         x.printStackTrace();
376     }
377 }
378
379 return bufferSound.toByteArray();
380 }
381
382 private static void clipping(int[] samples) {
383     int max16 = (int)(Math.pow(2, 15)) - 1;
384     int min16 = -(int)(Math.pow(2, 15));
385     for (int j = 0; j < samples.length; j++) {
386         if (samples[j] > max16)
387             samples[j] = max16;
388         else if (samples[j] < min16)
389             samples[j] = min16;
390     }
391 }
392 }

```

3.3 Obd.java

```

1 package applications;
2 import java.io.BufferedReader;
3 import java.io.File;
4 import java.io.FileOutputStream;
5 import java.io.IOException;
6 import java.io.InputStream;
7 import java.io.InputStreamReader;
8 import java.io.OutputStream;
9 import java.io.OutputStreamWriter;
10 import java.net.DatagramPacket;
11 import java.net.DatagramSocket;
12 import java.net.InetAddress;
13 import java.net.Socket;
14 import java.nio.charset.StandardCharsets;
15
16 public class Obd {
17
18     private static String[] header = {"01 1F", "01 0F", "01 11",
19                                     "01 0C", "01 0D", "01 05"};
20
21     public static void udpTelemetry(DatagramSocket socket,
22                                     InetAddress hostAddress, int serverPort,
23                                     String requestCode) {
24
25         byte[] rxbuffer = new byte[16];
26         DatagramPacket receivePacket =
27             new DatagramPacket(rxbuffer, rxbuffer.length);
28
29         for (int i = 0; i < header.length; i++) {
30             // TX
31             String completeCode = (requestCode + "OBD=" + header[i]);
32             byte[] txbuffer = completeCode.getBytes();
33             DatagramPacket sendPacket = new DatagramPacket(txbuffer, txbuffer.length,
34                                                             hostAddress, serverPort);
35             System.out.println("Complete request: " + completeCode);
36             try {
37                 socket.send(sendPacket);
38             } catch (Exception x) {

```

```

36     x.printStackTrace();
37 }
38 long timeBefore = System.currentTimeMillis();
39
40 // RX
41 try {
42     socket.setSoTimeout(3000);
43     socket.receive(receivePacket);
44     String message = new String(rxbuffer, StandardCharsets.US_ASCII);
45     System.out.println("Ithaki responded via UDP with: " + message);
46     System.out.println("Ithaki UDP time response: " +
47         (System.currentTimeMillis() - timeBefore) /
48         (float)1000 +
49         " seconds");
50
51     int[] values = parser(message);
52     formula(values[0], values[1], header[i]);
53 } catch (Exception x) {
54     x.printStackTrace();
55 }
56 }
57 }
58
59 public static float tcpTelemetry(Socket socket, FileWriter writerVehicle) {
60     float engineTime = 0;
61
62     try {
63         InputStream in = socket.getInputStream();
64         OutputStream out = socket.getOutputStream();
65         BufferedReader bf = new BufferedReader(new InputStreamReader(in));
66
67         for (int i = 0; i < header.length; i++) {
68             out.write((header[i] + "\r").getBytes());
69             // out.flush();
70             long timeBefore = System.currentTimeMillis();
71             System.out.println(
72                 "Created TCP socket and set output stream... Waiting for response");
73
74             System.out.println("Header: " + header[i]);
75             String data = bf.readLine();
76             System.out.println("Ithaki responded via TCP with: " + data);
77             System.out.println("Ithaki TCP time response: " +
78                 (System.currentTimeMillis() - timeBefore) /
79                 (float)1000 +
80                 " seconds");
81
82             int[] values = parser(data);
83             float value = formula(values[0], values[1], header[i]);
84             writerVehicle.write(value + " ");
85             if (header[i] == "01 1F")
86                 engineTime = value;
87         }
88
89         writerVehicle.write("\n");
90     } catch (Exception x) {
91         x.printStackTrace();
92     }
93
94     return engineTime;
95 }

```

```

96 private static float formula(int first, int second, String header) {
97     float value = 0;
98     switch (header) {
99         case "01 1F":
100             int engineRunTime = first * 256 + second;
101             System.out.println("Engine run time: " + engineRunTime);
102             value = engineRunTime;
103             break;
104
105         case "01 0F":
106             int intakeAirTemp = first - 40;
107             System.out.println("Intake Air Temperature: " + intakeAirTemp);
108             value = intakeAirTemp;
109             break;
110
111         case "01 11":
112             float throttlePos = (first * 100) / (float)255;
113             System.out.println("Throttle position: " + throttlePos);
114             value = throttlePos;
115             break;
116
117         case "01 0C":
118             float engineRpm = ((first * 256) + second) / (float)4;
119             System.out.println("Engine RPM: " + engineRpm);
120             value = engineRpm;
121             break;
122
123         case "01 0D":
124             int speed = first;
125             System.out.println("Vehicle speed: " + speed);
126             value = speed;
127             break;
128
129         case "01 05":
130             int coolantTemp = first - 40;
131             System.out.println("Coolant Temperature: " + coolantTemp);
132             value = coolantTemp;
133             break;
134
135         default:
136             System.out.println(
137                 "Something went wrong calculating formual for vehicle stats");
138     }
139     System.out.println();
140     return value;
141 }
142
143 private static int[] parser(String data) {
144     String byte1 = data.substring(6, 8);
145     int first = Integer.parseInt(byte1, 16);
146     String byte2 = "";
147     int second = 0;
148     if (data.length() > 8) {
149         byte2 = data.substring(9, 11);
150         second = Integer.parseInt(byte2, 16);
151     }
152     System.out.println();
153
154     int[] temp = {first, second};
155 }

```

```

156     return temp;
157 }
158 }

```

3.4 Copter.java

```

1 package applications;
2
3 import java.io.BufferedReader;
4 import java.io.ByteArrayOutputStream;
5 import java.io.FileWriter;
6 import java.io.InputStream;
7 import java.io.InputStreamReader;
8 import java.io.OutputStream;
9 import java.net.DatagramPacket;
10 import java.net.DatagramSocket;
11 import java.net.InetAddress;
12 import java.net.Socket;
13 import java.nio.charset.StandardCharsets;
14
15 public class Copter {
16     public static String udpTelemetry(DatagramSocket socket,
17                                     InetAddress hostAddress, int serverPort,
18                                     FileWriter writerCopter) {
19         // TX
20         // open ithakicopter.jar
21
22         // RX only
23         byte[] rxbuffer = new byte[128];
24         DatagramPacket receivePacket =
25             new DatagramPacket(rxbuffer, rxbuffer.length);
26
27         String telemetry = "";
28         try {
29             socket.setSoTimeout(3000);
30             socket.receive(receivePacket);
31             telemetry = new String(rxbuffer, StandardCharsets.US_ASCII);
32             System.out.println("Received data via UDP: " + telemetry);
33
34             String[] tokensMotor = telemetry.split("LMOTOR=");
35             String[] tokensAltitude = telemetry.split("ALTITUDE=");
36             String[] tokensTemp = telemetry.split("TEMPERATURE=");
37             String[] tokensPress = telemetry.split("PRESSURE=");
38             writerCopter.write(tokensMotor[1].substring(0, 3) + " ");
39             writerCopter.write(tokensAltitude[1].substring(0, 3) + " ");
40             writerCopter.write(tokensTemp[1].substring(1, 6) + " ");
41             writerCopter.write(tokensPress[1].substring(0, 7) + "\n");
42         } catch (Exception x) {
43             x.printStackTrace();
44         }
45         return telemetry;
46     }
47
48     public static String tcpTelemetry(InetAddress hostAddress, int target) {
49         String telemetry = "";
50         try (Socket socket = new Socket(hostAddress, 38048)) {
51             InputStream in = socket.getInputStream();
52             OutputStream out = socket.getOutputStream();
53             BufferedReader bf = new BufferedReader(new InputStreamReader(in));
54             ByteArrayOutputStream bos = new ByteArrayOutputStream();
55

```

```

56 String command = "AUTO FLIGHTLEVEL=" + target + " LMOTOR=" + target +
57                 " RMOTOR=" + target + " PILOT \r\n";
58 // System.out.print("Request: " + command);
59 out.write(command.getBytes());
60 out.flush();
61
62 // skip telemetry info
63 for (int i = 0; i < 14; i++) {
64     bos.write((bf.readLine() + "\n").getBytes());
65 }
66 String data = new String(bos.toByteArray(), StandardCharsets.US_ASCII);
67 // System.out.println("Received data via TCP: " + data);
68
69 String[] tokens = data.split("\n");
70 // take only the useful data and skip the info ithaki sent
71 telemetry = tokens[13];
72 } catch (Exception x) {
73     x.printStackTrace();
74 }
75 return telemetry;
76 }
77
78 /*
79  * tcpTelemetry function for the TX and udpTelemetry for RX. The way that
80  * these two functions are implemented force the autopilot to be used with a
81  * combination of these two. We want to send a command only if it is needed
82  * and we want to listen all the time to get feedback.
83  *
84  * Notes: Work In Progress
85  */
86 public static void autopilot(DatagramSocket listen, InetAddress hostAddress,
87                             int serverPort, Socket send, int lowerBound,
88                             int higherBound) {
89
90     lowerBound = Math.min(lowerBound, higherBound);
91     higherBound = Math.max(lowerBound, higherBound);
92
93     int target = (lowerBound + higherBound) / 2;
94     int motor = -1;
95
96     try {
97         System.out.println("AUTOPILOT: ON");
98         System.out.println(
99             "You need to open ithakicopter.jar. Press ENTER to continue...");
100        System.in.read();
101        System.out.println("Press Control-C to exit...");
102        Thread.sleep(1000);
103        for (;;) {
104            if ((motor < (lowerBound)) || (motor > (higherBound))) {
105                System.out.println("Send packet. Readjust...");
106                tcpTelemetry(hostAddress, target);
107            }
108
109            String telemetry =
110                Copter.udpTelemetry(listen, hostAddress, serverPort, null);
111            String[] tokens = telemetry.split("LMOTOR=");
112            motor = Integer.parseInt(tokens[1].substring(0, 3)); // get motor values
113
114            System.out.println("Parsed motor values: " + motor);
115        }

```



```
116     } catch (Exception x) {  
117         x.printStackTrace();  
118         System.out.println("AUTOPILOT failed");  
119     }  
120 }  
121 }
```