

```

1  /*
2  * Doinakis Michail 9292
3  * e-mail: doinakis@eceauth.gr
4  */
5  package com.javasSerialCommunications;
6
7  import ithakimodem.*;
8  import java.io.*;
9  import java.text.SimpleDateFormat;
10 import java.util.ArrayList;
11 import java.util.Date;
12 import java.util.List;
13
14 public class virtualModem {
15     /**
16      * Static variables for handling the modem
17      */
18     static Modem ithaki = new Modem();
19     static int modemSpeed = 1000;
20     static int modemSpeedImage = 80000;
21     static String modemName = "ithaki";
22     static int timeout = 2000;
23     static int expTime = 4;
24     static String folderLocation = "./";
25
26     /**
27      * Experiment Codes
28      */
29     static String echoCode = "E2640";
30
31     static String imageCode = "M0960";
32     static String cam = "FIX"; // or CAM = "FIX" or "PTZ" or ""
33     // Only for PTZ type of img.If CAM = "FIX" they are ignored
34     static String dir = "L";
35     static String size = "S";
36
37     static String imageCodeErrors = "G9493";
38     static String camErrors = "PTZ";
39     // Only for PTZ type of img.If CAM = "FIX" they are ignored
40     static String dirErrors = "L";
41     static String sizeErrors = "S";
42
43     static String gpsCode = "P0348";
44     static String gpsRoute = "1015099";
45     static int numberOfMarks = 9;
46     static int timeBetweenMarks = 10;
47
48     static String ackCode = "Q6936";
49     static String nackCode = "R8684";
50
51     public static void main(String[] args) throws IOException {
52         (new virtualModem()).demo();
53     }
54
55     /**
56      * Performs all the experiments for the given assignment
57      * @throws IOException throws IO exception if there is an error creating the file
58      */
59     public void demo() throws IOException {
60         SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd 'at' HH:mm:ss z");
61         Date date;
62         ithaki.setSpeed(modemSpeed);
63         ithaki.setTimeout(timeout);
64         openModem(ithaki, modemName);
65
66         /*

```

```

67      * Echo packet response times experiment
68      */
69      date = new Date(System.currentTimeMillis());
70      System.out.println("Echo Packet experiment started: " + formatter.format(date));
71
72      echoPacketResponseTime(ithaki,echoCode + "\r",expTime);
73
74      date = new Date(System.currentTimeMillis());
75      System.out.println("Echo Packet experiment ended: " + formatter.format(date));
76
77      /*
78      * Image request experiment
79      */
80      // Error free
81      ithaki.setSpeed(modemSpeedImage);
82      String imgLocation = folderLocation + imageCode + ".jpg";
83      date = new Date(System.currentTimeMillis());
84      System.out.println("Requesting error free image: " + formatter.format(date));
85      getImage(ithaki,imageCode,cam,dir,size,imgLocation);
86      date = new Date(System.currentTimeMillis());
87      System.out.println("Image received: " + formatter.format(date));
88
89      // With errors
90      imgLocation = folderLocation + imageCodeErrors + ".jpg";
91      date = new Date(System.currentTimeMillis());
92      System.out.println("Requesting image with errors: " + formatter.format(date));
93      getImage(ithaki,imageCodeErrors,camErrors,dirErrors,sizeErrors,imgLocation);
94      date = new Date(System.currentTimeMillis());
95      System.out.println("Image received: " + formatter.format(date));
96
97      /*
98      * Gps request experiment
99      */
100     List<String> R = new ArrayList<>();
101     R.add(gpsRoute);
102     imgLocation = folderLocation + "gpsImage.jpg";
103     date = new Date(System.currentTimeMillis());
104     System.out.println("Requesting GPS route image: " + formatter.format(date));
105     getGPSTime(ithaki,gpsCode,R,imgLocation,numberOfMarks,timeBetweenMarks);
106     date = new Date(System.currentTimeMillis());
107     System.out.println("Image received: " + formatter.format(date));
108
109     /*
110     * Automatic repeat request
111     */
112     ithaki.setSpeed(modemSpeed);
113     date = new Date(System.currentTimeMillis());
114     System.out.println("Automatic Repeat experiment started: " + formatter.format(date));
115     arqPacketExperiment(ithaki,ackCode + "\r",nackCode + "\r",expTime);
116     date = new Date(System.currentTimeMillis());
117     System.out.println("Automatic Repeat Request experiment ended: " + formatter.format
118     (date));
119
120     ithaki.close();
121 }
122 /**
123  * Method that initializes a connection with the virtual modem
124  * @param modem a modem class
125  * @param modemName the name of the modem to connect to (In this case ithaki)
126  */
127 static void openModem(Modem modem,String modemName) {
128     try{
129         if(!modem.open(modemName)) throw new CustomExceptionMessage("Could not open to
modem.");
130         printHelloMessage(modem);

```

```

130         }catch(Exception e){
131             System.out.println(e);
132         }
133     }
134
135     /**
136      * Method that prints the Greetings message that ithaki modems send at first connection
with it
137      * @param modem a modem class
138      */
139     static void printHelloMessage(Modem modem) {
140         int characterReceived, counter=0;
141         char[] endSequence = {'\r', '\n', '\n', '\n'};
142         do {
143             try{
144                 characterReceived = modem.read();
145
146                 if ((char)characterReceived == endSequence[counter]) counter += 1;
147                 else counter = 0;
148
149                 if(characterReceived == -1) throw new CustomExceptionMessage("Modem
disconnected");
150                 System.out.print((char)characterReceived);
151             }catch (Exception e){
152                 System.out.println(e);
153                 return;
154             }
155         }while(counter != endSequence.length);
156     }
157
158     /**
159      * Method that calculates the response times of the ithaki server in a certain period
of time
160      * @param modem a modem class
161      * @param echoCode the echo code for the particular date and time provided by ithaki
lab
162      * @param time how long the experiment will continue asking ithaki server for echo
packets (in minutes)
163      */
164     static void echoPacketResponseTime(Modem modem, String echoCode, int time) {
165         List<Long> responseTimes = new ArrayList<>();
166         long timeElapsed, totalTime=0L, experimentTime=(long)time*60000;
167         while(totalTime < experimentTime) {
168             timeElapsed = System.currentTimeMillis();
169             responseTimes.add(getEchoPacket(modem, echoCode));
170             timeElapsed = System.currentTimeMillis() - timeElapsed;
171             totalTime += timeElapsed;
172         }
173         StringBuilder toWriteEchoResponseTimes = new StringBuilder();
174         for (Long responseTime : responseTimes) {
175             toWriteEchoResponseTimes.append(responseTime).append(",");
176         }
177         try {
178             File myFile1 = new File(folderLocation + "echoExperiment.csv");
179             Writer writer = new PrintWriter(myFile1);
180             writer.write(toWriteEchoResponseTimes.toString());
181             writer.close();
182         } catch (Exception e) {
183             System.out.println(e);
184         }
185     }
186
187     /**
188      * Method that requests a single echo packet from the ithaki server
189      * @param modem a modem class
190      * @param echoCode the echo code for the particular date and time provided by ithaki

```

```

190 lab
191     * @return the response time of a single packet
192     */
193     static long getEchoPacket(Modem modem, String echoCode) {
194
195         long responseTime=0L;
196         char[] startSequence = "PSTART".toCharArray();
197         char[] stopSequence = "PSTOP".toCharArray();
198         int characterReceived,stopCounter=0,iterationCounter=0;
199         boolean startCorrect=true;
200         try{
201             if(!modem.write(echoCode.getBytes()))
202                 throw new CustomExceptionMessage("Could not request packet from server.");
203             responseTime = System.currentTimeMillis();
204
205         }catch (Exception e){
206             System.out.println(e);
207             System.exit(1);
208         }
209         do{
210             try{
211                 characterReceived = modem.read();
212                 if (characterReceived == -1) throw new CustomExceptionMessage("Modem
disconnected during packet request");
213                 if ((char) characterReceived == stopSequence[stopCounter]) stopCounter += 1
;
214                 else stopCounter = 0;
215                 if (iterationCounter < startSequence.length){
216                     if (characterReceived != startSequence[iterationCounter]) startCorrect
= false;
217                     if (!startCorrect) throw new CustomExceptionMessage("Unexpected packet
format");
218                     iterationCounter++;
219                 }
220                 if (stopCounter == stopSequence.length){
221                     responseTime = System.currentTimeMillis() - responseTime;
222                 }
223             }catch (Exception e){
224
225                 System.out.println(e);
226                 System.exit(1);
227             }
228         }while(stopCounter != stopSequence.length);
229         return responseTime;
230     }
231
232     /**
233     * Method that receives a requested image from the server and saves it locally
234     * @param modem a modem class
235     * @param imgCode the requested code
236     * @param imgLocation the location to store the image
237     * @throws IOException throws IO exception if there is an error creating the file
238     */
239     static void requestImage(Modem modem, String imgCode, String imgLocation) throws
IOException {
240         boolean startCorrect=true;
241         int characterReceived,stopCounter=0,iterationCounter=0;
242         int[] startSequence = {255,216};
243         int[] endSequence = {255,217};
244         File image = new File(imgLocation);
245         FileOutputStream fos = new FileOutputStream(image);
246         try{
247             if (!modem.write(imgCode.getBytes()))
248                 throw new CustomExceptionMessage("Could not request image from server.");
249         }catch (Exception e){
250             System.out.println(e);

```

```

251         System.exit(1);
252     }
253     do{
254         try{
255             characterReceived = modem.read();
256             if (characterReceived == -1) throw new CustomExceptionMessage("Modem
disconnected during image request");
257             if (characterReceived == endSequence[stopCounter]) stopCounter += 1;
258             else stopCounter = 0;
259             fos.write((byte) characterReceived);
260             if(iterationCounter < startSequence.length){
261                 if(characterReceived != startSequence[iterationCounter]) startCorrect
= false;
262                 if(!startCorrect) throw new CustomExceptionMessage("Unexpected image
format");
263                 iterationCounter++;
264             }
265         }catch (Exception e){
266             System.out.println(e);
267             System.exit(1);
268         }
269         if (stopCounter == endSequence.length) {
270             fos.close();
271         }
272     }while(stopCounter != endSequence.length);
273 }
274
275 /**
276  * Method that requests and saves an image requested from the ithaki server
277  * @param modem          a modem class
278  * @param imageCode      the image code for the particular date and time provided by
ithaki lab
279  * @param cam            parameter for which camera to be used
280  * @param dir            direction of the camera dir = "R" or "L" or "U" or "D"(right,
left,up,down)(applies only for cam = "PTZ")
281  * @param size          size of the requested image size = "L" or "R" (applies only for
cam = "PTZ")
282  * @param imgLocation   the location to store the image
283  * @throws IOException  throws IO exception if there is an error creating the file
284  */
285 static void getImage(Modem modem,String imageCode,String cam,String dir,String size,
String imgLocation) throws IOException {
286
287     imageCode = constructImageCode(imageCode,cam,dir,size);
288     requestImage(modem, imageCode, imgLocation);
289 }
290
291 /**
292  * Method that constructs an image code given the CAM,DIR,SIZE parameters
293  * @param imageCode the requested image code
294  * @param cam        the code of the camera
295  * @param dir        the direction (L,R,U,D)
296  * @param size       the desirable size of the image (S,L)
297  * @return          returns a string with the code and the desirable image parameters
298  */
299 static String constructImageCode(String imageCode,String cam,String dir,String size){
300     boolean bool = dir.equals("L") || dir.equals("U") || dir.equals("R") || dir.equals(
"D");
301     switch(cam) {
302         case "PTZ":
303             cam = "CAM=PTZ";
304             if(bool) dir = "DIR=" + dir;
305             else dir = "";
306             if(size.equals("S") || size.equals("L")) size = "SIZE=" + size;
307             else size = "";
308             break;

```

```

309         case "FIX":
310             cam = "CAM=FIX";
311             dir = "";
312             size = "";
313             break;
314         default:
315             cam = "CAM=" + cam;
316             if(bool) dir = "DIR=" + dir;
317             else dir = "";
318             if((size.equals("S") || size.equals("L"))) size = "SIZE=" + size;
319             else size = "";
320             break;
321     }
322     imageCode = imageCode + cam + dir + size + "\r";
323
324     return imageCode;
325 }
326
327 /**
328  * Method that requests gps marks from ithaki server
329  * @param modem a modem class
330  * @param gpsCode the requested gps code
331  * @param R route parameters
332  * @throws IOException throws IO exception if there is an error creating the file
333  */
334 static void getGPSTrack(Modem modem,String gpsCode,List<String> R,String imgLocation,int
335 numberOfMarks,int timeBetweenMarks) throws IOException {
336     char[] startSequence = "START ITHAKI GPS TRACKING\r\n".toCharArray();
337     char[] stopSequence = "STOP ITHAKI GPS TRACKING\r\n".toCharArray();
338     String gpsMarkCode = constructGPSTrack(gpsCode,R,true);
339     int characterReceived,stopCounter=0,iterationCounter=0;
340     boolean startCorrect=true;
341     try{
342         if (!modem.write(gpsMarkCode.getBytes()))
343             throw new CustomExceptionMessage("Could not request packet from server.");
344     }catch (Exception e){
345         System.out.println(e);
346         System.exit(1);
347     }
348     String gpsMark = "";
349     do{
350         try{
351             characterReceived = modem.read();
352             if (characterReceived == -1) throw new CustomExceptionMessage("Modem
353 disconnected during packet request");
354             if ((char) characterReceived == stopSequence[stopCounter]) stopCounter += 1
355 ;
356             else stopCounter = 0;
357             gpsMark += (char) characterReceived;
358             if(iterationCounter < startSequence.length) {
359                 if(characterReceived != startSequence[iterationCounter]) startCorrect
360 = false;
361                 if(!startCorrect) throw new CustomExceptionMessage("Unexpected packet
362 format");
363                 iterationCounter++;
364             }
365         }catch (Exception e){
366             System.out.println(e);
367             System.exit(1);
368         }
369     }while(stopCounter != stopSequence.length);
370     gpsMark = gpsMark.substring(startSequence.length,gpsMark.length()-stopSequence.
371 length);
372     String latitude;

```

```

369     String longitude;
370     List<String> T = new ArrayList<>();
371     int secondsLat,secondsLon;
372     int k = gpsMark.split("\r\n").length;
373     int i = 0;
374     double prevTime = 0.0;
375     double currTime;
376     double time;
377     String[] markSplit;
378     String test;
379     for(int c = 0;c < k;c++) {
380         markSplit = gpsMark.split("\r\n")[c].split(",");
381         currTime = Double.parseDouble(markSplit[1].substring(0,2)) * 3600 + Double.
parseDouble(markSplit[1].substring(2,4))* 60 + Double.parseDouble(markSplit[1].substring(4
));
382         time = currTime - prevTime;
383         if(time >= timeBetweenMarks && i < numberOfMarks) {
384             latitude = markSplit[2];
385             longitude = markSplit[4];
386             secondsLat = (int)Math.round(Double.parseDouble(latitude.substring(4)) * 60
);
387             secondsLon = (int)Math.round(Double.parseDouble(longitude.substring(5)) *
60);
388             test = longitude.substring(1,5) + secondsLon + latitude.substring(0,4) +
secondsLat;
389             if(!T.contains(test)) {
390                 T.add(test);
391                 i++;
392             }
393             prevTime = currTime;
394         }
395     }
396     String gpsImgCode = constructGPSCode(gpsCode,T,false);
397     requestImage(modem,gpsImgCode,imgLocation);
398 }
399 }
400
401 /**
402  * Method that constructs a gps request code
403  * @param gpsCode the requested gps code
404  * @param R gps marks from a certain route (e.g R="XPPPLL") or gps marks jpeg
image (e.g T="AABBCCDDEEZZ")
405  * @param type if type is true then parameter R is included in the code, otherwise
R is a list with marks for the image
406  * @return returns a gps code either requesting image with marks on it or just
gps marks
407  */
408 static String constructGPSCode(String gpsCode,List<String> R,boolean type) {
409     if(type) {
410         if (!R.isEmpty()) {
411             gpsCode = gpsCode + "R=" + R.get(0);
412         }
413     }else {
414         if (!R.isEmpty()) {
415             for (String s : R) {
416                 gpsCode = gpsCode + "T=" + s;
417             }
418         }
419     }
420     gpsCode = gpsCode + "\r";
421     return gpsCode;
422 }
423
424 /**
425  * Method that performs the ARQ packet experiment
426  * @param modem a modem class

```



```

427 * @param ackCode request code that indicates that the packets arrived correctly
428 * @param nackCode request code that indicates that the packets arrived incorrectly
429 * @param time the time the experiment will tun
430 */
431 static void arqPacketExperiment(Modem modem,String ackCode,String nackCode,int time) {
432     List<Integer> numberOfNack = new ArrayList<>();
433     List<Long> packetResponseTime = new ArrayList<>();
434     long timeElapsed,totalTime=0L,experimentTime=(long)time*60000;
435
436     while(totalTime < experimentTime) {
437         timeElapsed = System.currentTimeMillis();
438         numberOfNack.add(getCorrectPacket(modem,ackCode,nackCode));
439         timeElapsed = System.currentTimeMillis() - timeElapsed;
440         packetResponseTime.add(timeElapsed);
441         totalTime += timeElapsed;
442     }
443     String toWriteARQTimes="";
444     StringBuilder toWriteNumberOfARQ = new StringBuilder();
445     for (Long aLong : packetResponseTime) {
446         toWriteARQTimes += aLong + ",";
447     }
448     for (Integer integer : numberOfNack) {
449         toWriteNumberOfARQ.append(integer).append(",");
450     }
451     try {
452         File myFile1 = new File(folderLocation + "ArqResponseTimes.csv");
453         File myFile2= new File(folderLocation + "ArqNumberOfNack.csv");
454         Writer writer1 = new PrintWriter(myFile1);
455         Writer writer2 = new PrintWriter(myFile2);
456         writer1.write(toWriteARQTimes);
457         writer2.write(toWriteNumberOfARQ.toString());
458         writer1.close();
459         writer2.close();
460     } catch (Exception e) {
461         System.out.println(e);
462     }
463 }
464
465 /**
466  * Method that counts how many times a specific packet is requested
467  * @param modem a modem class
468  * @param ackCode the code that requests the next packet if the received packet is
correct
469  * @param nackCode the code that requests the same packet if its received incorrectly
470  * @return returns the number of times a packet its requested
471  */
472 static int getCorrectPacket(Modem modem,String ackCode,String nackCode) {
473     int numberOfNack=0;
474     if(!requestARQCode(modem,ackCode)) {
475         numberOfNack++;
476         while (!requestARQCode(modem,nackCode)) {
477             numberOfNack++;
478         }
479     }
480     return numberOfNack;
481 }
482
483 /**
484  * Method that requests a packet from the server
485  * @param modem a modem class
486  * @param arqCode the request code (either ACK or Nack)
487  * @return returns true if the requested packet arrives correctly, false otherwise
488  */
489 static boolean requestARQCode(Modem modem,String arqCode) {
490     char[] startSequence = "PSTART".toCharArray();
491     char[] stopSequence = "PSTOP".toCharArray();

```



```

492     int characterReceived, stopCounter=0, iterationCounter=0;
493     boolean startCorrect=true;
494     String arqResponse = "";
495     try{
496         if (!modem.write(arqCode.getBytes()))
497             throw new CustomExceptionMessage("Could not request packet from server.");
498     }
499     catch (Exception e){
500         System.out.println(e);
501         System.exit(1);
502     }
503     do{
504         try{
505             characterReceived = modem.read();
506             if (characterReceived == -1) throw new CustomExceptionMessage("Modem
disconnected during packet request");
507             if ((char) characterReceived == stopSequence[stopCounter]) stopCounter += 1
;
508             else stopCounter = 0;
509             arqResponse += (char)characterReceived;
510             if(iterationCounter < startSequence.length) {
511                 if(characterReceived != startSequence[iterationCounter]) startCorrect
= false;
512                 if(!startCorrect) throw new CustomExceptionMessage("Unexpected packet
format");
513                 iterationCounter++;
514             }
515             }catch (Exception e) {
516                 System.out.println(e);
517                 System.exit(1);
518             }
519         }while(stopCounter != stopSequence.length);
520
521         char[] coded = arqResponse.split(" ")[4].substring(1,17).toCharArray();
522         int fcs = Integer.parseInt(arqResponse.split(" ")[5]);
523         int codedFCS = 0;
524         for (char c : coded) {
525             codedFCS = codedFCS ^ (int) c;
526         }
527
528         return (codedFCS == fcs);
529     }
530 }
531 }
532
533 /**
534  * Custom class to throw custom exceptions
535  */
536 class CustomExceptionMessage extends Exception {
537     public CustomExceptionMessage(String message) {
538         super(message);
539     }
540 }
541

```