```java
/*
 * Doinakis Michail 9292
 * e-mail: doinakis@eceauth.gr
 */
package com.javasSerialCommunications;

import ithakimodem.*;
import java.io.*;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

public class Main {

    public static void main(String[] args) throws IOException{
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd 'at' HH:mm:ss z");
        Date date;

        Modem modem = new Modem(2000);
        String modemName = "ithaki";
        modem.setTimeout(2000);
        openModem(modem,modemName);

        int expTime = 4;
        /*
         * Echo packet response times experiment
         */
        String echoCode = "E4511\r";
        getEchoPacket(modem,echoCode);
        getEchoPacket(modem,echoCode);
        date = new Date(System.currentTimeMillis());
        System.out.println("Echo Packet experiment started: " + formatter.format(date));

        echoPacketResponseTime(modem,echoCode,expTime);

        date = new Date(System.currentTimeMillis());
        System.out.println("Echo Packet experiment ended: " + formatter.format(date));
        /*
         * Image request experiment
         */
        // Error free
        modem.setSpeed(80000);
        String imageCode = "M9556";
        String cam = "PTZ"; // or CAM = "FIX" or "PTZ" or ""
        String dir = "";
        String size = "";
        String imgLocation = "./session/imgPTZErrorFree.jpg";
        constructImageCode(imageCode,cam,dir,size);
        date = new Date(System.currentTimeMillis());
        System.out.println("Requesting error free image: " + formatter.format(date));
        getImage(modem,imageCode,cam,dir,size,imgLocation);
        date = new Date(System.currentTimeMillis());
        System.out.println("Image  received: " + formatter.format(date));

        // With errors
        imageCode = "G7481";
        cam = "PTZ";
        dir = "";
        size = "";
        imgLocation = "./session/imgPTZErrors.jpg";
        date = new Date(System.currentTimeMillis());
        System.out.println("Requesting image with errors: " + formatter.format(date));
        getImage(modem,imageCode,cam,dir,size,imgLocation);
        date = new Date(System.currentTimeMillis());
        System.out.println("Image  received: " + formatter.format(date));
```

```java
        /*
         * Gps request experiment
         */
        String gpsCode = "P5712";
        List<String> R = new ArrayList<>();
        R.add("1015099");
        imgLocation = "./session/gpsImage.jpg";
        int numberOfMarks = 9;
        int timeBetweenMarks = 10;
        date = new Date(System.currentTimeMillis());
        System.out.println("Requesting GPS route image: " + formatter.format(date));
        getGPSMark(modem,gpsCode,R,imgLocation,numberOfMarks,timeBetweenMarks);
        date = new Date(System.currentTimeMillis());
        System.out.println("Image  received: " + formatter.format(date));

        /*
         * Automatic repeat request
         */
        modem.setSpeed(2000);
        String ackCode = "Q9912\r";
        String nackCode = "R1580\r";
        date = new Date(System.currentTimeMillis());
        System.out.println("Automatic Repeat experiment started: " + formatter.format(date
));
        arqPacketExperiment(modem,ackCode,nackCode,expTime);
        date = new Date(System.currentTimeMillis());
        System.out.println("Automatic Repeat Request experiment ended: " + formatter.format
(date));

        modem.close();

    }

    /**
     * Method that initializes a connection with the virtual modem
     * @param modem a modem class
     * @param modemName the name of the modem to connect to (In this case ithaki)
     */
    public static void openModem(Modem modem,String modemName){
        try{
            if(!modem.open(modemName)) throw new customExceptionMessage("Could not open to
modem.");
            printHelloMessage(modem);
        }catch(Exception e){
            System.out.println(e);
        }
    }

    /**
     * Method that prints the Greetings message that ithaki modems send at first connection
 with it
     * @param modem a modem class
     */
    public static void printHelloMessage(Modem modem){
        int characterReceived,counter=0;
        char[] endSequence = {'\r','\n','\n','\n'};
        do {
            try{
                characterReceived = modem.read();

                if ((char)characterReceived == endSequence[counter]) counter += 1;
                else counter = 0;

                if(characterReceived == -1) throw new customExceptionMessage("Modem
disconnected");
                System.out.print((char)characterReceived);
```

```
128             }catch (Exception e){
129                 System.out.println(e);
130                 return;
131             }
132         }while(counter != endSequence.length);
133     }
134 
135     /**
136      * Method that calculates the response times of the ithaki server in a certain period
    of time
137      * @param modem a modem class
138      * @param echoCode the echo code for the particular date and time provided by ithaki
    lab
139      * @param time how long the experiment will continue asking ithaki server for echo
    packets (in minutes)
140      */
141     public static void echoPacketResponseTime(Modem modem,String echoCode,int time){
142         List<Long> responseTimes = new ArrayList<>();
143         long timeElapsed,totalTime=0L,experimentTime=(long)time*60000;
144         while(totalTime < experimentTime){
145             timeElapsed = System.currentTimeMillis();
146             responseTimes.add(getEchoPacket(modem,echoCode));
147             timeElapsed = System.currentTimeMillis() - timeElapsed;
148             totalTime += timeElapsed;
149         }
150         String toWriteEchoResponseTimes = "";
151         for (Long responseTime : responseTimes) {
152             toWriteEchoResponseTimes += responseTime + ",";
153         }
154         try {
155             File myFile1 = new File("./session/echoExperiment.csv");
156             Writer writer = new PrintWriter(myFile1);
157             writer.write(toWriteEchoResponseTimes);
158             writer.close();
159         } catch (Exception e) {
160             System.out.println(e);
161         }
162     }
163 
164     /**
165      * Method that requests a single echo packet from the ithaki server
166      * @param modem a modem class
167      * @param echoCode the echo code for the particular date and time provided by ithaki
    lab
168      * @return the response time of a single packet
169      */
170     public static long getEchoPacket(Modem modem, String echoCode){
171 
172         long responseTime=0L;
173         char[] startSequence = "PSTART".toCharArray();
174         char[] stopSequence = "PSTOP".toCharArray();
175         int characterReceived,stopCounter=0,iterationCounter=0;
176         boolean startCorrect=true;
177         try{
178             if(!modem.write(echoCode.getBytes()))
179                 throw new customExceptionMessage("Could not request packet from server.");
180             responseTime = System.currentTimeMillis();
181 
182         }catch (Exception e){
183             System.out.println(e);
184             System.exit(1);
185         }
186         do{
187             try{
188                 characterReceived = modem.read();
189                 if (characterReceived == -1) throw new customExceptionMessage("Modem
```

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

```
248             }
249         }while(stopCounter != endSequence.length);
250     }
251 
252     /**
253      * Method that requests and saves an image requested from the ithaki server
254      * @param modem         a modem class
255      * @param imageCode     the image code for the particular date and time provided by
    ithaki lab
256      * @param cam           parameter for which camera to be used
257      * @param dir           direction of the camera dir = "R" or "L" or "U" or "D"(right,
    left,up,down)(applies only for cam = "PTZ")
258      * @param size          size of the requested image size = "L" or "R" (applies only for
     cam = "PTZ")
259      * @param imgLocation   the location to store the image
260      * @throws IOException  throws IO exception if there is an error creating the file
261      */
262     public static void getImage(Modem modem,String imageCode,String cam,String dir,String
    size,String imgLocation) throws IOException {
263 
264         imageCode = constructImageCode(imageCode,cam,dir,size);
265         requestImage(modem, imageCode, imgLocation);
266     }
267 
268     /**
269      * Method that constructs an image code given the CAM,DIR,SIZE parameters
270      * @param imageCode the requested image code
271      * @param cam       the code of the camera
272      * @param dir       the direction (L,R,U,D)
273      * @param size      the desirable size of the image (S,L)
274      * @return          returns a string with the code and the desirable image parameters
275      */
276     public static String constructImageCode(String imageCode,String cam,String dir,String
    size){
277         boolean bool = dir.equals("L") || dir.equals("U") || dir.equals("R") || dir.equals(
    "D");
278         switch(cam){
279             case "PTZ":
280                 cam = "CAM=PTZ";
281                 if(bool) dir = "DIR=" + dir;
282                 else dir = "";
283                 if(size.equals("S") || size.equals("L")) size = "SIZE=" + size;
284                 else size = "";
285                 break;
286             case "FIX":
287                 cam = "CAM=FIX";
288                 dir = "";
289                 size = "";
290                 break;
291             default:
292                 cam = "CAM=" + cam;
293                 if(bool) dir = "DIR=" + dir;
294                 else dir = "";
295                 if((size.equals("S") || size.equals("L"))) size = "SIZE=" + size;
296                 else size = "";
297                 break;
298         }
299         imageCode = imageCode + cam + dir + size + "\r";
300 
301         return imageCode;
302     }
303 
304     /**
305      *
306      * @param modem         a modem class
307      * @param gpsCode       the requested gps code
```

```
308      * @param R             route parameters
309      * @throws IOException  throws IO exception if there is an error creating the file
310      */
311     public static void getGPSMark(Modem modem,String gpsCode,List<String> R,String
    imgLocation,int numberOfMarks,int timeBetweenMarks) throws IOException {
312 
313         char[] startSequence = "START ITHAKI GPS TRACKING\r\n".toCharArray();
314         char[] stopSequence = "STOP ITHAKI GPS TRACKING\r\n".toCharArray();
315         String gpsMarkCode = constructGPSCode(gpsCode,R,true);
316         int characterReceived,stopCounter=0,iterationCounter=0;
317         boolean startCorrect=true;
318         try{
319             if (!modem.write(gpsMarkCode.getBytes()))
320                 throw new customExceptionMessage("Could not request packet from server.");
321         }catch (Exception e){
322             System.out.println(e);
323             System.exit(1);
324         }
325         String gpsMark = "";
326         do{
327             try{
328                 characterReceived = modem.read();
329                 if (characterReceived == -1) throw new customExceptionMessage("Modem
    disconnected during packet request");
330                 if ((char) characterReceived == stopSequence[stopCounter]) stopCounter += 1
    ;
331                 else stopCounter = 0;
332                 gpsMark += (char) characterReceived;
333                 if(iterationCounter < startSequence.length){
334                     if(characterReceived != startSequence[iterationCounter]) startCorrect
     = false;
335                     if(!startCorrect) throw new customExceptionMessage("Unexpected packet
    format");
336                     iterationCounter++;
337                 }
338             }catch (Exception e){
339                 System.out.println(e);
340                 System.exit(1);
341             }
342         }while(stopCounter != stopSequence.length);
343 
344         gpsMark = gpsMark.substring(startSequence.length,gpsMark.length()-stopSequence.
    length);
345         List<String> latitude = new ArrayList<>();
346         List<String> longitude = new ArrayList<>();
347         List<String> T = new ArrayList<>();
348         int secondsLat,secondsLon;
349         int k = gpsMark.split("\r\n").length;
350         int i = 0;
351         double prevTime = 0.0;
352         double currTime;
353         double time;
354         String[] markSplit;
355         String test;
356         for(int c = 0;c < k;c++){
357             markSplit = gpsMark.split("\r\n")[c].split(",");
358             currTime = Double.parseDouble(markSplit[1].substring(0,2)) * 3600 + Double.
    parseDouble(markSplit[1].substring(2,4))* 60 + Double.parseDouble(markSplit[1].substring(4
    ));
359             time = currTime - prevTime;
360             if(time >= timeBetweenMarks && i < numberOfMarks){
361                 latitude.add(markSplit[2]);
362                 longitude.add(markSplit[4]);
363                 secondsLat = (int)Math.round(Double.parseDouble(latitude.get(i).substring(4
    )) * 60);
364                 secondsLon = (int)Math.round(Double.parseDouble(longitude.get(i).substring(
```

```
364 5)) * 60);
365                 test = longitude.get(i).substring(1,5) + secondsLon + latitude.get(i).
    substring(0,4) + secondsLat;
366                 if(!T.contains(test)) {
367                     T.add(test);
368                     i++;
369                 }else{
370                     latitude.remove(i);
371                     longitude.remove(i);
372                 }
373                 prevTime = currTime;
374             }
375         }
376         String gpsImgCode = constructGPSCode(gpsCode,T,false);
377         requestImage(modem,gpsImgCode,imgLocation);
378 
379     }
380 
381     /**
382      * Method that constructs a gps request code
383      * @param gpsCode   the requested gps code
384      * @param R         gps marks from a certain route (e.g R="XPPPLL") or gps marks jpeg
    image (e.g T="AABBCCDDEEZZ")
385      * @param type      if type is true then parameter R is included in the code, otherwise
     R is a list with marks for the image
386      * @return          returns a gps code either requesting image with marks on it or just
     gps marks
387      */
388     public static String constructGPSCode(String gpsCode,List<String> R,boolean type){
389         if(type){
390             if (!R.isEmpty()) {
391                 gpsCode = gpsCode + "R=" + R.get(0);
392             }
393         }else{
394             if (!R.isEmpty()) {
395                 for (String s : R) {
396                     gpsCode = gpsCode + "T=" + s;
397                 }
398             }
399         }
400         gpsCode = gpsCode + "\r";
401         return gpsCode;
402     }
403 
404     /**
405      * Method that performs the ARQ packet experiment
406      * @param modem     a modem class
407      * @param ackCode   request code that indicates that the packets arrived correctly
408      * @param nackCode  request code that indicates that the packets arrived incorrectly
409      * @param time      the time the experiment will tun
410      */
411     public static void arqPacketExperiment(Modem modem,String ackCode,String nackCode,int
    time){
412         List<Integer> numberOfNack = new ArrayList<>();
413         List<Long> packetResponseTime = new ArrayList<>();
414         long timeElapsed,totalTime=0L,experimentTime=(long)time*60000;
415 
416         while(totalTime < experimentTime){
417             timeElapsed = System.currentTimeMillis();
418             numberOfNack.add(getCorrectPacket(modem,ackCode,nackCode));
419             timeElapsed = System.currentTimeMillis() - timeElapsed;
420             packetResponseTime.add(timeElapsed);
421             totalTime += timeElapsed;
422         }
423         String toWriteARQTimes="";
424         String toWriteNumberOfARQ = "";
```

```
425         for (Long aLong : packetResponseTime) {
426             toWriteARQTimes += aLong + ",";
427         }
428         for (Integer integer : numberOfNack) {
429             toWriteNumberOfARQ += integer + ",";
430         }
431         try {
432             File myFile1 = new File("./session/ArqResponseTimes.csv");
433             File myFile2= new File("./session/ArqNumberOfNack.csv");
434             Writer writer1 = new PrintWriter(myFile1);
435             Writer writer2 = new PrintWriter(myFile2);
436             writer1.write(toWriteARQTimes);
437             writer2.write(toWriteNumberOfARQ);
438             writer1.close();
439             writer2.close();
440         } catch (Exception e) {
441             System.out.println(e);
442         }
443     }
444 
445     /**
446      * Method that counts how many times a specific packet is requested
447      * @param modem     a modem class
448      * @param ackCode   the code that requests the next packet if the received packet is
    correct
449      * @param nackCode  the code that requests the same packet if its received incorrectly
450      * @return  returns the number of times a packet its requested
451      */
452     public static int getCorrectPacket(Modem modem,String ackCode,String nackCode){
453         int numberOfNack=0;
454         if(!requestARQCode(modem,ackCode)) {
455             numberOfNack++;
456             while (!requestARQCode(modem,nackCode)) {
457                 numberOfNack++;
458             }
459         }
460         return numberOfNack;
461     }
462 
463     /**
464      * Method that requests a packet from the server
465      * @param modem     a modem class
466      * @param arqCode   the request code (either ACK or Nack)
467      * @return returns true if the requested packet arrives correctly, false otherwise
468      */
469     public static boolean requestARQCode(Modem modem,String arqCode){
470         char[] startSequence = "PSTART".toCharArray();
471         char[] stopSequence = "PSTOP".toCharArray();
472         int characterReceived,stopCounter=0,iterationCounter=0;
473         boolean startCorrect=true;
474         String arqResponse = "";
475         try{
476             if (!modem.write(arqCode.getBytes()))
477                 throw new customExceptionMessage("Could not request packet from server.");
478 
479         }catch (Exception e){
480             System.out.println(e);
481             System.exit(1);
482         }
483         do{
484             try{
485                 characterReceived = modem.read();
486                 if (characterReceived == -1) throw new customExceptionMessage("Modem
    disconnected during packet request");
487                 if ((char) characterReceived == stopSequence[stopCounter]) stopCounter += 1
    ;
```

```
488                 else stopCounter = 0;
489                 arqResponse += (char)characterReceived;
490                 if(iterationCounter < startSequence.length){
491                     if(characterReceived != startSequence[iterationCounter]) startCorrect
     = false;
492                     if(!startCorrect) throw new customExceptionMessage("Unexpected packet
    format");
493                     iterationCounter++;
494                 }
495             }catch (Exception e){
496 
497                 System.out.println(e);
498                 System.exit(1);
499             }
500         }while(stopCounter != stopSequence.length);
501 
502         char[] coded = arqResponse.split(" ")[4].substring(1,17).toCharArray();
503         int fcs = Integer.parseInt(arqResponse.split(" ")[5]);
504         int codedFCS = 0;
505         for (char c : coded) {
506             codedFCS = codedFCS ^ (int) c;
507         }
508 
509         return (codedFCS == fcs);
510     }
511 }
512 
513 /**
514  * Custom class to throw custom exceptions
515  */
516 class customExceptionMessage extends Exception {
517     public customExceptionMessage(String message){
518         super(message);
519     }
520 }
521 
```