```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 virtualModem {
    /**
     * Static variables for handling the modem
     */
    static Modem modem = new Modem();
    static int modemSpeed = 1000;
    static int modemSpeedImage = 80000;
    static String modemName = "ithaki";
    static int timeout = 2000;
    static int expTime = 4;
    static String folderLocation = "./session/";

    /**
     * Experiment Codes
     */
    static String echoCode = "E7745";

    static String imageCode = "M7525";
    static String cam = "FIX"; // or CAM = "FIX" or "PTZ" or ""
    // Only for PTZ type of img.If CAM = "FIX" they are ignored
    static String dir = "L";
    static String size = "S";

    static String imageCodeErrors = "G2202";
    static String camErrors = "PTZ";
    // Only for PTZ type of img.If CAM = "FIX" they are ignored
    static String dirErrors = "L";
    static String sizeErrors = "S";

    static String gpsCode = "P1108";
    static String gpsRoute = "1015099";
    static int numberOfMarks = 9;
    static int timeBetweenMarks = 10;

    static String ackCode = "Q8694";
    static String nackCode = "R8666";

    public static void main(String[] args) throws IOException {
        (new virtualModem()).demo();
    }

    /**
     * Performs all the experiments for the given assignment
     * @throws IOException throws IO exception if there is an error creating the file
     */
    public void demo() throws IOException {
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd 'at' HH:mm:ss z");
        Date date;
        modem.setTimeout(timeout);
        openModem(modem,modemName);

        /*
         * Echo packet response times experiment
```

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

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

```java
     * @return the response time of a single packet
     */
    static long getEchoPacket(Modem modem, String echoCode) {

        long responseTime=0L;
        char[] startSequence = "PSTART".toCharArray();
        char[] stopSequence = "PSTOP".toCharArray();
        int characterReceived,stopCounter=0,iterationCounter=0;
        boolean startCorrect=true;
        try{
            if(!modem.write(echoCode.getBytes()))
                throw new CustomExceptionMessage("Could not request packet from server.");
            responseTime = System.currentTimeMillis();

        }catch (Exception e){
            System.out.println(e);
            System.exit(1);
        }
        do{
            try{
                characterReceived = modem.read();
                if (characterReceived == -1) throw new CustomExceptionMessage("Modem disconnected during packet request");
                if ((char) characterReceived == stopSequence[stopCounter]) stopCounter += 1;
                else stopCounter = 0;
                if (iterationCounter < startSequence.length){
                    if (characterReceived != startSequence[iterationCounter]) startCorrect = false;
                    if (!startCorrect) throw new CustomExceptionMessage("Unexpected packet format");
                    iterationCounter++;
                }
                if (stopCounter == stopSequence.length){
                    responseTime = System.currentTimeMillis() - responseTime;
                }
            }catch (Exception e){

                System.out.println(e);
                System.exit(1);
            }
        }while(stopCounter != stopSequence.length);
        return responseTime;
    }

    /**
     * Receives a requested image from the server
     * @param modem         a modem class
     * @param imgCode       the requested code
     * @param imgLocation   the location to store the image
     * @throws IOException  throws IO exception if there is an error creating the file
     */
    static void requestImage(Modem modem, String imgCode, String imgLocation) throws IOException {
        boolean startCorrect=true;
        int characterReceived,stopCounter=0,iterationCounter=0;
        int[] startSequence = {255,216};
        int[] endSequence = {255,217};
        File image = new File(imgLocation);
        FileOutputStream fos = new FileOutputStream(image);
        try{
            if (!modem.write(imgCode.getBytes()))
                throw new CustomExceptionMessage("Could not request image from server.");
        }catch (Exception e){
            System.out.println(e);
            System.exit(1);
```

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

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

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

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

```
492         char[] startSequence = "PSTART".toCharArray();
493         char[] stopSequence = "PSTOP".toCharArray();
494         int characterReceived,stopCounter=0,iterationCounter=0;
495         boolean startCorrect=true;
496         String arqResponse = "";
497         try{
498             if (!modem.write(arqCode.getBytes()))
499                 throw new CustomExceptionMessage("Could not request packet from server.");
500 
501         }catch (Exception e){
502             System.out.println(e);
503             System.exit(1);
504         }
505         do{
506             try{
507                 characterReceived = modem.read();
508                 if (characterReceived == -1) throw new CustomExceptionMessage("Modem
    disconnected during packet request");
509                 if ((char) characterReceived == stopSequence[stopCounter]) stopCounter += 1
    ;
510                 else stopCounter = 0;
511                 arqResponse += (char)characterReceived;
512                 if(iterationCounter < startSequence.length) {
513                     if(characterReceived != startSequence[iterationCounter]) startCorrect
     = false;
514                     if(!startCorrect) throw new CustomExceptionMessage("Unexpected packet
    format");
515                     iterationCounter++;
516                 }
517             }catch (Exception e) {
518 
519                 System.out.println(e);
520                 System.exit(1);
521             }
522         }while(stopCounter != stopSequence.length);
523 
524         char[] coded = arqResponse.split(" ")[4].substring(1,17).toCharArray();
525         int fcs = Integer.parseInt(arqResponse.split(" ")[5]);
526         int codedFCS = 0;
527         for (char c : coded) {
528             codedFCS = codedFCS ^ (int) c;
529         }
530 
531         return (codedFCS == fcs);
532     }
533 }
534 
535 /**
536  * Custom class to throw custom exceptions
537  */
538 class CustomExceptionMessage extends Exception {
539     public CustomExceptionMessage(String message) {
540         super(message);
541     }
542 }
543 
```