



ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

(6^ο εξάμηνο)

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

Εργασία δικτυακού προγραμματισμού
JAVA SERIAL COMMUNICATIONS PROGRAMMING

SOURCE CODE

Το αποθετήριο της εργασίας βρίσκεται στο GitHub, κάνοντας *click* στο παρακάτω κουμπί:



[<https://github.com/Kyparissis/Networks1-2022-Assignment>]

Για οποιαδήποτε διευκρίνιση ή απορία για την εργασία αυτή ή εντοπίσετε κάποιο λάθος,
παρακαλώ επικοινωνήστε μαζί μου!

⇒ Ακαδημαϊκό e-mail: kyparkypar@ece.auth.gr

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμo: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[2]









Η αναφορά αυτή περιέχει τον **πηγαίο κώδικα της εφαρμογής *userApplication***.

Επίσης περιέχει τον πηγαίο κώδικα του *package applications* που περιέχει τα τέσσερα (4) *.java files* με τις μεθόδους που χρησιμοποιούνται για κάθε ζητούμενο / *application* {*Echo*, *ARQ*, *Image*, *GPS*} από την *userApplication*. Τέλος περιέχεται και ο τρόπος που είναι δομημένος ο πηγαίος κώδικας του *Java project* της *userApplication*.

(Προτείνεται το κατέβασμα και η χρήση του πηγαίου κώδικα από το repository στο *GitHub*)

(Η δημιουργία και η εκτέλεση του κώδικα έγινε με την χρήση του *Eclipse IDE*)

ΔΟΜΗ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ

 <i>Networks1-2022-Assignment</i>	
 <i>lib</i>	
 <i>ithakimodem.jar</i>	
 <i>src</i>	3
 <i>UserApplication.java</i>	3
 <i>applications</i>	8
 <i>Echo.java</i>	8
 <i>ARQ.java</i>	10
 <i>Image.java</i>	14
 <i>GPS.java</i>	16

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[3]



UserApplication.java

```
import applications.*;

import ithakimodem.*;

//-----
//      LIBRARIES
//-----
import java.io.File;
import java.io.FileWriter;
import java.time.LocalDateTime;           // To Import the LocalDateTime class
import java.time.format.DateTimeFormatter; // To Import the DateTimeFormatter class

/**
 * @apiNote This is a SCRIPT that runs the applications needed to "communicate" receive/generate
 * output requested from the assignment instructions (Assignment-Instructions.pdf) .
 *
 * (From each session, we keep/save the output folder)
 */
public class UserApplication {

    public static void main(String[] args) {

        System.out.println("Computer Networks I - ASSIGNMENT");
        System.out.println("JAVA SERIAL COMMUNICATIONS PROGRAMMING");
        System.out.println("-----");

        // -----
        //      SAVING Ithaki's REQUEST CODES into VARIABLES
        //      (from: http://ithaki.eng.auth.gr/netlab/action.php)
        // -----
        final String echo_request_code = "E????";           // Echo request code
        final String image_request_code_withoutErrors = "M????"; // Image request code (Tx/Rx error free)
        final String image_request_code_withErrors = "G????"; // Image request code (Tx/Rx with errors)
        final String gps_request_code = "P????";           // GPS request code
        final String ack_result_code = "Q????";            // ARQ - ACK (positive acknowledgement) result code
        final String nack_result_code = "R????";           // ARQ - NACK (negative acknowledgement) result code

        // -----
        //      CREATING the virtual modem && SETTINGS
        // -----
        Modem IthakiModem = new Modem();

        int IthakiModemSPEED = 80000;
        IthakiModem.setSpeed(IthakiModemSPEED);           // SETTING the virtual modem's (Ithaki's) SPEED [Limits: 1 - 80 Kbps]
        int IthakiModemTIMEOUT = 2000;
        IthakiModem.setTimeout(IthakiModemTIMEOUT);       // SETTING the virtual modem's (Ithaki's) TIMEOUT

        // =====
        //      OPENING the virtual modem
        // =====
        if(IthakiModem.open("ithaki")) { // "ithaki" argument opens up an Ithaki's Virtual Modem (Other modems available)
            // Console output:
            System.out.println();
            System.out.println("=====");
            System.out.println("      Opened an Ithaki's virtual modem!      ");
        }
    }
}
```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμo: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[4]

```
        System.out.println();
        System.out.println("Modem's SPEED: " + IthakiModemSPEED + "bps           ");
        System.out.println("Modem's TIMEOUT: " + IthakiModemTIMEOUT + " (s)       ");
        System.out.println("=====");
    } else {
        return;
    }

    // CURRENTLY IN "COMMAND" MODE.
    // Only "AT" (from "AT"ention) type of commands are accepted

    // -----
    //           Setting the virtual modem to data mode
    //           (NEEDED TO RECEIVE EVERY TYPE OF COMMAND)
    // -----
    // Console output:
    System.out.println();
    System.out.println("=====");
    System.out.println("  DIALING UP Ithaki - Setting the modem to DATA mode ");
    System.out.println("    (Needed to receive every other type of command)  ");
    System.out.println();
    System.out.println("           Typing \"ATD2310ITHAKI\" ...           ");
    System.out.println("=====");
    //
    if(IthakiModem.write("ATD2310ITHAKI\r".getBytes())) {
        getALLOutputLines(IthakiModem);
    }

    // -----
    //           Testing the connection to the modem by typing TEST
    // -----
    // Console output:
    System.out.println();
    System.out.println("=====");
    System.out.println(" Typing \"TEST\" to test the connection the modem... ");
    System.out.println("=====");
    //
    if(IthakiModem.write(("TEST\r").getBytes())) {
        String TESTOutputLines = getALLOutputLines(IthakiModem);
        // Expected output: PSTART DD-MM-YYYY HH:MM:SS ITHAKI JAVA LAB SERVER TEST PSTOP
        if (!TESTOutputLines.contains("ITHAKI JAVA LAB SERVER TEST")) {
            // Console output:
            System.out.println("Connection couldn't be tested. Aborting...");

            return;
        }
    }
}

// -----
//           APPLICATIONS' RESULTS and MEASUREMENTS
// -----
// Console output:
System.out.println();
System.out.println("=====");
System.out.println("           Running ALL the APPLICATIONS...           ");
System.out.println("=====");
// An array containing date and times the application started running
LocalDateTime[] DateTime = new LocalDateTime[5];

/*
 * -----
 * |      Echo Application      |
 * -----
 */
// Console output:
System.out.println();
System.out.println("=====");
System.out.println("|      Echo Application      |");
System.out.println("=====");
System.out.println();
```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[5]

```
DateTime[0] = LocalDateTime.now(); // Time the Echo application starts
//
Echo.EchoApplication(IthakiModem, echo_request_code, (5 * 60), "output/Echo/");

/* -----
 * |      ARQ Application      |
 * -----
 */
// Console output:
System.out.println();
System.out.println("-----");
System.out.println("|      ARQ Application      |");
System.out.println("-----");
System.out.println();
DateTime[1] = LocalDateTime.now(); // Time the ARQ application starts
//
ARQ.ARQApplication(IthakiModem, ack_result_code, nack_result_code, (5 * 60), "output/ARQ/");

/* -----
 * |  CAM-IMAGES Application  |
 * -----
 */
// Console output:
System.out.println();
System.out.println("-----");
System.out.println("|  CAM-IMAGES Application  |");
System.out.println("-----");
System.out.println();
DateTime[2] = LocalDateTime.now(); // Time the Images application starts
//
// (image_request_code_withoutErrors produces an error free image)
// (image_request_code_withErrors produces pseudoerrors in the image)
// When "CAM=FIX" (Default value) accompanies the image_request_code, the fixxed camera is used
// When "CAM=PTZ" accompanies the image_request_code, the moveable (using a servo moter) camera is used
Image.generateImage(Image.receiveImageBytes(IthakiModem, image_request_code_withoutErrors + "CAM=FIX"),
    "output/images/error-free/image_errorFree_FIX.jpg");
Image.generateImage(Image.receiveImageBytes(IthakiModem, image_request_code_withoutErrors + "CAM=PTZ"),
    "output/images/error-free/image_errorFree_PTZ.jpg");
Image.generateImage(Image.receiveImageBytes(IthakiModem, image_request_code_withErrors + "CAM=FIX"),
    "output/images/with-errors/image_withErrors_FIX.jpg");
Image.generateImage(Image.receiveImageBytes(IthakiModem, image_request_code_withErrors + "CAM=PTZ"),
    "output/images/with-errors/image_withErrors_PTZ.jpg");

/* -----
 * |      GPS Application      |
 * -----
 */
// Console output:
System.out.println();
System.out.println("-----");
System.out.println("|      GPS Application      |");
System.out.println("-----");
System.out.println();
DateTime[3] = LocalDateTime.now(); // Time the GPS application starts
//
// When "R=XPPPPLL" accompanies the code gps_request_code, Ithaki's server send LL GPS packets from
// the pre-saved route X, starting with the packet PPPP.
// HERE: Route -> X = 1
// Starting packet -> PPPP = 0280 (Random packet number we chose)
// Number of packets -> LL = 99 (MAX)
String RParameter = "R=1028099";
String FULL_TParameter = GPS.GPSApplication(IthakiModem, gps_request_code + RParameter);
// Returns the: FULL_TParameter = "T=AABCCDDEEFF" (Longitude: AA BB' CC' and Latitude: DD EE' FF')
//
// Generate image taken, from Google Maps at the cordinates with Longitude: AA BB' CC' & Latitude: DD EE' FF', by Ithaki
System.out.println("(Receiving Google Maps image with the chosen points being marked...)");
System.out.println(".");
System.out.print(".");
DateTime[4] = LocalDateTime.now(); // Time the GPS-IMAGE application starts
Image.generateImage(Image.receiveImageBytes(IthakiModem, gps_request_code + FULL_TParameter),
```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[6]

```
        "output/images/GPS/GPS_Image_GoogleMaps.jpg");
// -----

// =====
//          Closed the Ithaki's virtual modem!
// =====
if(IthakiModem.close()) {
    // Console output:
    System.out.println();
    System.out.println("=====");
    System.out.println("          Closed the Ithaki's virtual modem!          ");
    System.out.println("=====");
} else
    return;

// Files output:
saveAppsDatesTimes2FILE(DateTime);
saveRequestCodes2FILE(echo_request_code, image_request_code_withoutErrors, image_request_code_withErrors,
    gps_request_code, ack_result_code, nack_result_code, RParameter , FULL_TParameter);
}

/**
 * Reads the VirtualModem's output and return the full text it returns
 * (Reads until -1 is returned from the modem (End of stream))
 *
 * @param VirtualModem Ithaki's modem we send the request to
 *
 * @return WHOLE text from the VirtualModem's output
 */
public static String getALLOutputLines(Modem VirtualModem) {
    int returnIntModem;
    String FullText = "";
    while (true) {
        returnIntModem = VirtualModem.read();
        if (returnIntModem == -1) {
            break;
        }
        System.out.print((char) returnIntModem);
        FullText += (char) returnIntModem;
    }
    return FullText;
}

/**
 * Saves all the request codes used into a .txt file
 */
private static void saveRequestCodes2FILE(String echo_request_code,String image_request_code_withoutErrors,
    String image_request_code_withErrors, String gps_request_code, String ack_result_code,
    String nack_result_code, String RParameter, String FULL_TParameter){
    try(FileWriter RequestCodes = new FileWriter(new File("output/" + "RequestCodes.txt"))){
        RequestCodes.write("Echo request code: " + echo_request_code + "\r\n");
        RequestCodes.write("ARQ request code: " + image_request_code_withoutErrors + "\r\n");
        RequestCodes.write("Image request code (Tx/Rx error free): " + image_request_code_withoutErrors + "\r\n");
        RequestCodes.write("Image request code (Tx/Rx with errors): " + image_request_code_withErrors + "\r\n");

        RequestCodes.write("GPS request code: " + gps_request_code + "\r\n");
        RequestCodes.write("GPS - R= Parameter code: " + RParameter + "\r\n");
        RequestCodes.write("GPS - Points' Full T= Parameter code: " + FULL_TParameter + "\r\n");
        RequestCodes.write("ARQ - ACK (positive acknowledgement) result code: " + ack_result_code + "\r\n");

        RequestCodes.write("ARQ - NACK (negative acknowledgement) result code: " + nack_result_code + "\r\n");

    } catch (Exception x) {
        System.out.println("Caught EXCEPTION: " + x + " ! Couldn't create files: output/RequestCodes.txt .");
        System.out.println("Request codes are not saved into a file!");
    }
}

/**
 * Saves the date and time the applications started into a .txt file
 */
}
```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμo: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[7]

```
*
* @param DatesTimes An array containing the date and times the application started running
*/
private static void saveAppsDatesTimes2FILE(LocalDateTime[] DatesTimes) {
    try(FileWriter AppsExecutionDatesTimes = new FileWriter(new File("output/" + "AppsExecutionDatesTimes.txt"))){
        DateTimeFormatter DateTimeFORMAT = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");

        AppsExecutionDatesTimes.write("Echo application run at: " + DatesTimes[0].format(DateTimeFORMAT) + "\r\n");
        AppsExecutionDatesTimes.write("ARQ application run at: " + DatesTimes[1].format(DateTimeFORMAT) + "\r\n");
        AppsExecutionDatesTimes.write("Images application run at: " + DatesTimes[2].format(DateTimeFORMAT) + "\r\n");
        AppsExecutionDatesTimes.write("GPS application run at: " + DatesTimes[3].format(DateTimeFORMAT) + "\r\n");
        AppsExecutionDatesTimes.write("GPS-IMAGE application run at: " + DatesTimes[3].format(DateTimeFORMAT) + "\r\n");

    } catch (Exception x) {
        System.out.println("Caught EXCEPTION: " + x + " ! Couldn't create files: output/AppsExecutionDatesTimes.txt .");
        System.out.println("Applications' execution times are not saved into a file!");
    }
}
```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[8]



applications



Echo.java

```
package applications;

import ithakimodem.*;

// -----
//          LIBRARIES
// -----

import java.io.File;
import java.io.FileWriter;

/**
 * @apiNote This class contains methods in order to receive text packet's
 * from Ithaki's VirtualModem
 */
public class Echo {
    /**
     * This method receives ECHO packets from Ithaki's VirtualModem for the duration
     * of the runtimeInSeconds (sec). Then saves in a file the time it took to receive
     * that packet from the VirtualModem.
     * (Those files are saved in the folder directory [Use \ for UNIX, / for WINDOWS])
     *
     * @param VirtualModem Ithaki's modem we send the request to
     * @param EchoRequestCode The request code in order to receive Echo packets
     * @param runtimeInSeconds Duration of the application
     * @param folder The directory to store the output files
     *
     * @return
     */
    public static void EchoApplication(Modem VirtualModem, String EchoRequestCode, int runtimeInSeconds, String folder) {
        // Trying to create output files in order to store: ECHO packet times
        try (FileWriter echoPacketsTimes = new FileWriter(new File(folder + "Echo_PacketsReceiveTimesMillis.txt"))) {
            String EchoPacketText = "";

            // ECHO packet TIMERS
            long packet_sentTime = 0; // The time the echo packet was sent from the modem
            long packet_receivedTime = 0; // The time we received the echo packet from the modem

            // COUNTERS
            int NumberOfPacketsReceived = 0;

            long startCallingPacketsTime = System.currentTimeMillis(); // Time we started to ask for packets from the modem

            while (System.currentTimeMillis() <= startCallingPacketsTime + 1000 * runtimeInSeconds) {

                // Receiving packets and calculating it's time:
                packet_sentTime = System.currentTimeMillis();
                EchoPacketText = receivePacket(VirtualModem, EchoRequestCode);
                packet_receivedTime = System.currentTimeMillis();

                // Increasing counters accordingly
                NumberOfPacketsReceived++;

                // Console output:
                System.out.println("-> Received Echo packet: " + EchoPacketText);
                System.out.println("[ECHO][Packet #" + NumberOfPacketsReceived + " (PC=" + getPacketPC(EchoPacketText)
```


ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[9]

```
        + ") delivery time: " + (packet_receivedTime - packet_sentTime) + " (ms)]\n");
```

```
        // Files output:
```

```
        echoPacketsTimes.write((packet_receivedTime - packet_sentTime) + "\r\n");    // Save packet's response time
```

```
    }
```

```
//    long stopCallingPacketsTime = System.currentTimeMillis(); // Time we stopped asking for echo packets from the modem
```

```
    // Results console output:
```

```
    System.out.println("[ECHO] Received " + NumberOfPacketsReceived
```

```
        + " packets SUCCESSFULLY from Ithaki's Virtual Modem ]");
```

```
    } catch (Exception x) {
```

```
        System.out.println("Caught EXCEPTION: " + x + " ! Couldn't create files: " + folder + "echoPacketsTimes.txt AND" +  
                            folder + "Echo_PacketsINFO.txt .");
```

```
    }
```

```
}
```

```
/**
```

```
 * This method sends a request to Ithaki's VirtualModem using the
```

```
 * RequestCode and receives as an "answer" a String packet.
```

```
 *
```

```
 * @param VirtualModem    Ithaki's modem we send the request to
```

```
 * @param RequestCode     The request code in order to receive ECHO packets
```

```
 *
```

```
 * @return Packet's text FULL String
```

```
 */
```

```
private static String receivePacket(Modem VirtualModem, String RequestCode) {
```

```
    String ReceivedPacketText = ""; // Variable to build the packet's string
```

```
    // Sending request to the virtual modem.
```

```
    boolean VirtualModemResponse = VirtualModem.write((RequestCode + "\r").getBytes());
```

```
    // Getting virtual modem's "answer" to the request.
```

```
    if (VirtualModemResponse) {
```

```
        // Every packet starts with header PSTART and ends with footer PSTOP
```

```
        // We can until our "builder" string ends with the footer
```

```
        int VirtualModemOutput;
```

```
        while (!ReceivedPacketText.endsWith("PSTOP")) {
```

```
            VirtualModemOutput = VirtualModem.read(); // Read "answer's" Integer
```

```
            ReceivedPacketText += (char) VirtualModemOutput; // Append Integer's char value to packet's String
```

```
        }
```

```
    }
```

```
    return ReceivedPacketText;
```

```
}
```

```
/**
```

```
 * Methods to parse the "Echo type" packet sent by the VirtualModem
```

```
 *
```

```
 * @param EchoPacket : An "Echo type" packet sent by the VirtualModem. Example:
```

```
 *
```

```
        "PSTART DD-MM-YYYY HH-MM-SS PC PSTOP"
```

```
 */
```

```
// -----
```

```
// Returns DATE (from given ex. = "DD-MM-YYYY")
```

```
// private static String getPacketDate(String EchoPacket) {
```

```
//     return EchoPacket.split(" ")[1];
```

```
// }
```

```
// Returns HOUR (from given ex. = "HH-MM-SS")
```

```
// private static String getPacketHour(String EchoPacket) {
```

```
//     return EchoPacket.split(" ")[2];
```

```
// }
```

```
// Returns PacketCounter%100 (from given ex. = "PC")
```

```
private static String getPacketPC(String EchoPacket) {
```

```
    return EchoPacket.split(" ")[3];
```

```
}
```

```
// -----
```

```
}
```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[10]



ARQ.java

```
package applications;

import ithakimodem.*;

//-----
//          LIBRARIES
//-----
import java.io.File;
import java.io.FileWriter;

/**
 * @apiNote This class contains methods in order to receive text packet's from Ithaki's
 * VirtualModem and cope with possible errors in them using ARQ (Automatic Repeat Request)
 * mechanisms.
 */
public class ARQ {

    /**
     * This method uses an Automatic Repeat Request method to receive ARQ packets
     * from Ithaki's VirtualModem for the duration of the runtimeInSeconds (sec)
     * Then saves in a file the number of nack request each packet (ack request) needed AND in a
     * second file saves the time it took to receive that packet (without noise) from the VirtualModem.
     * (Those files are saved in the folder directory [Use \ for UNIX, / for WINDOWS])
     * At last, it calculates the Bit Error Rate using the calculateBER() method ...
     *
     * @param VirtualModem          Ithaki's modem we send the request to
     * @param AckResult_RequestCode ACK(positive acknowledgement) result packet request code (sends NEW packet)
     * @param NackResult_RequestCode NACK(negative acknowledgement) result packet request code (sends same packet)
     * @param runtimeInSeconds      Duration of the ARQ mechanism
     * @param folder                The directory to store the output files
     */
    public static void ARQApplication(Modem VirtualModem, String AckResult_RequestCode, String NackResult_RequestCode, int runtimeInSeconds, String folder) {
        // Trying to create output files in order to store: ARQ Packet times &&
        //                                     Nack requests per ARQ for a packet
        try (FileWriter ARQPacketsTimes = new FileWriter(new File(folder + "ARQ_PacketsReceiveTimesMillis.txt"));
            FileWriter ARQAckResultsCounterPerPacket = new FileWriter(new File(folder + "ARQ_AckResultsPerPacket.txt"));
            FileWriter ARQBitErrorRate = new FileWriter(new File(folder + "ARQ_BitErrorRate.txt"))) {

            String[] getPacketARQResults = new String[2];
            int NackResultsCounterPerARQPacket = 0;
            String ARQPacketText = "";

            // ARQ packet TIMERS
            long packet_sentTime = 0;           // Time we send the first request (Ack)
            long packet_receivedTime = 0; // Time we receive the correct packet after a number of Nack requests

            // Requests COUNTERS
            int NackResultsFULLCounter = 0;      // Each Packet means NackResultsCounterPerARQPacket results
            int AckResultsFULLCounter = 0;        // Each packet means 1 ACK result

            long startCallingPacketsTime = System.currentTimeMillis(); // Time we started to ask for echo packets from the modem

            // Running the ARQ Application for runtimeInSeconds (sec):
            while (System.currentTimeMillis() <= startCallingPacketsTime + 1000 * runtimeInSeconds) {

                // Receiving packets and calculating it's time:
                packet_sentTime = System.currentTimeMillis();           // Time we send the next packet request
                getPacketARQResults = receivePacket_StopAndWaitARQ(VirtualModem, AckResult_RequestCode, NackResult_RequestCode);
                packet_receivedTime = System.currentTimeMillis();        // Time we receive the packet (Error-FREE)
                //
                // AND:
                // Saving the results from the getPacketARQ(VirtualModem, AckResult_RequestCode, NackResult_RequestCode) into variables
                ARQPacketText = getPacketARQResults[0];
                NackResultsCounterPerARQPacket = Integer.valueOf(getPacketARQResults[1]);

                // Increasing ACK and NACK counters accordingly
                NackResultsFULLCounter += NackResultsCounterPerARQPacket;
                AckResultsFULLCounter++; // == Error-free packets we received

                // Console output:
                System.out.println("[ARQ][Packet #" + AckResultsFULLCounter + " (PC=" + getPacketPC(ARQPacketText) + ") delivery time: "

```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμo: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[11]

```

                                                                    + (packet_receivedTime - packet_sentTime) + " (ms)]\n");

//
// Files output:
ARQAckResultsCounterPerPacket.write(AckResultsCounterPerARQPacket + "\r\n"); // packet's ack counter
ARQPacketsTimes.write((packet_receivedTime - packet_sentTime) + "\r\n"); // packet's system response times
}

//
long stopCallingPacketsTime = System.currentTimeMillis(); // Time we stopped asking for echo packets from the modem

// Results console output:
System.out.println("[ARQ][ Received " + AckResultsFULLCounter + " packets SUCCESSFULLY (Error-FREE) from Ithaki's Virtual Modem ]");

System.out.println("\n( We had, IN TOTAL: "+ AckResultsFULLCounter + " NACK(negative acknowledgement) packet results in a total of "
    + (AckResultsFULLCounter + AckResultsFULLCounter) + " requests. )");
System.out.println(".");
double BER = getBitErrorRate(AckResultsFULLCounter, AckResultsFULLCounter);

// Console output:
System.out.println("BER (Bit Error Rate) Value = " + BER);
// File output:
ARQBitErrorRate.write("BER (Bit Error Rate) Value = " + BER);
} catch (Exception x) { // If files couldn't be created, THROW EXCEPTION
    System.out.println("Caught EXCEPTION: " + x + " ! Couldn't create files: " + folder + "ARQPacketsTimes.txt AND"
        + folder + "ARQPacketsAckRequestsPerPacket.txt .");
}

}

/**
 * Stop-and-wait ARQ Protocol Algorithm.
 * This method uses the AckResult_RequestCode to receive a new packet from Ithaki's VirtualModem.
 * (Positive ackn. since last packet was received successfully)
 * Then compares the packet's FCS with the packet's sequence's XOR result to determine
 * on whether this packet is error-free or not.
 * --> If ERRORS (SequenceXORResult != FCS): Resend the same packet using the NackResult_RequestCode and check again.
 * --> If Error-FREE (SequenceXORResult == FCS): return the error-free packet's text String
 *
 * @param VirtualModem Ithaki's modem we send the request to
 * @param AckResult_RequestCode ACK(positive acknowledgement) result packet request code (sends NEW packet)
 * @param NackResult_RequestCode NACK(negative acknowledgement) result packet request code (sends same packet)
 *
 * @return ARRAY String with: [0]: The error-free packet's text String
 * [1]: (String)The number of times we re-requested the same packet due to errors in it.
 */
public static String[] receivePacket_StopAndWaitARQ(Modem VirtualModem, String AckResult_RequestCode, String NackResult_RequestCode) {

    // Asking the modem to send a new packet.
    String ReceivedPacketText = receivePacket(VirtualModem, AckResult_RequestCode);

    // Console output:
    System.out.println("--> Received NEW packet, with ack_request_code: " + ReceivedPacketText);

    int SequenceXORResult = 0, FCS = 0;
    int AckResultsCounter = 0;
    while (true) {
        SequenceXORResult = getSequenceXOR(getPacketEncryptedSequence(ReceivedPacketText));
        FCS = getFCS_INT(ReceivedPacketText);

        // Checking for errors in last received packet
        if (SequenceXORResult != FCS) {
            // ERRORS FOUND!
            // Console output:
            System.out.println("---> Sequence XOR Result != FCS (ERRORS in packet). Requesting SAME packet with nack_request_code...");

            // Asking the modem to send the same packet again
            ReceivedPacketText = receivePacket(VirtualModem, NackResult_RequestCode);
            AckResultsCounter ++;

            // Console output:
            System.out.println("---> Received packet again. Now we got: " + ReceivedPacketText);
        } else {
            // ERROR FREE
            break;
        }
    }
    // Console output:
    System.out.println("-----> Finally, Error-FREE packet is: " + ReceivedPacketText );
    System.out.println("[ARQ][Received after " + AckResultsCounter + " NACK results / re-requests]");

    // Return the error free packet AND the number of times the modem resent the same packet due to errors

```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[12]

```
        return new String[]{ReceivedPacketText,
                             Integer.toString(NackResultsCounter));
    }

    /**
     * This method sends a request to Ithaki's VirtualModem using the
     * RequestCode and receives as an "answer" a String packet.
     *
     * @param VirtualModem      Ithaki's modem we send the request to
     * @param RequestCode       The request code in order to receive same or new packet
     *
     * @return Packet's text FULL String
     */
    private static String receivePacket(Modem VirtualModem, String RequestCode) {
        String ReceivedPacketText = ""; // Variable to build the packet's string

        // Sending request to the virtual modem.
        boolean VirtualModemResponse = VirtualModem.write((RequestCode + "\r").getBytes());
        // Getting virtual modem's "answer" to the request.
        if (VirtualModemResponse) {
            // Every packet starts with header PSTART and ends with footer PSTOP
            // We can until our "builder" string ends with the footer
            int VirtualModemOutput;
            while (!ReceivedPacketText.endsWith("PSTOP")) {
                VirtualModemOutput = VirtualModem.read(); // Read "answer's" Integer
                ReceivedPacketText += (char) VirtualModemOutput; // Append Integer's char value to packet's String
            }
            return ReceivedPacketText;
        }
    }

    /**
     * Calculates the result of applying the XOR operator consecutively at the 16
     * "pseudoencrypted" characters that are included in the ARQ packet.
     *
     * @param Sequence The "pseudoencrypted" characters String
     *
     * @return the final XOR result as an Integer
     */
    private static int getSequenceXOR(String Sequence) {
        char XORResult = 0; // Initialise to 0 because of the identity element ( A ^ 0 = A ).
        // ( This means that any value XOR'd with zero is left unchanged )
        // So FIRSTLY XORResult = (Sequence.toCharArray())[0] ^ 0 = (Sequence.toCharArray())[0]
        // and then XORResult = (Sequence.toCharArray())[0] ^ (Sequence.toCharArray())[1]
        // and then XORResult = ...
        // and
        for (int i = 0; i < (Sequence.toCharArray()).length; i++) {
            XORResult ^= (Sequence.toCharArray())[i];
        }

        return (int) XORResult;
    }

    /**
     * Calculates the BER (Bit Error Rate) by using the formula:
     *
     *  $P1 = (1 - Pb) ^ F$  (1)
     * ,where: Pb: Probability that a bit is received in error; aka the Bit Error Rate (BER)
     *         P1: Probability that a frame/sequence arrives with no bit errors
     *         F: Number of bits per frame/sequence
     *
     * so (1) =>  $P1 ^ (1/F) = 1 - Pb$  =>
     *           $Pb = 1 - P1 ^ (1/F)$  <=>
     *           $BER = 1 - P1 ^ (1/F)$ 
     *
     * @see Bibliography:
     * Data and Computer Communications 8th Edition William Starlings -> Page 186 (6.3 ERROR DETECTION)
     *
     * @param NumberOfFramesWITHOUTError The number of frames/sequences that had no bit errors
     * @param NumberOfFramesWITHError     The number of frames/sequences that had bit error(s)
     *
     * @return BER (Bit Error Rate) value
     */
    public static double getBitErrorRate(int NumberOfFramesWITHOUTError, int NumberOfFramesWITHError) {
        double BER = 0; // Probability that a bit is received in error.
        double P1 = 0; // Probability that a frame/sequence arrives with no bit errors
        double F = 0; // Number of bits per packet sequence
        int bytesPerChar = 2; // Char is 2 bytes in Java (Unicode)( Unlike C/C++ (1 byte in C - ASCII))
        int bitsPerByte = 8; // Each byte is equal to 8 bits
        int bitsPerChar = bytesPerChar * bitsPerByte;
        int CharactersPerPacket = 16;

        F = CharactersPerPacket * bitsPerChar;
    }
```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[13]

```
P1 = (((double)NumberOfFramesWITHOUTError) / ((double)(NumberOfFramesWITHOUTError + NumberOfFramesWITHError)));
BER = 1 - Math.pow(P1, (1.0/F));
```

```
return BER;
```

```
}
```

```
/**
```

```
 * Methods to parse the "ARQ type" packet sent by the VirtualModem
```

```
 *
```

```
 * @param ARQPacket : An "ARQ type" packet sent by the VirtualModem. Example:
```

```
 * "PSTART DD-MM-YYYY HH-MM-SS PC <XXXXXXXXXXXXXXXXX> FCS PSTOP"
```

```
 */
```

```
// -----
```

```
// Returns DATE (from given ex. = "DD-MM-YYYY")
```

```
private static String getPacketDate(String ARQPacket) {
```

```
    return ARQPacket.split(" ")[1];
```

```
}
```

```
// Returns HOUR (from given ex. = "HH-MM-SS")
```

```
private static String getPacketHour(String ARQPacket) {
```

```
    return ARQPacket.split(" ")[2];
```

```
}
```

```
// Returns PacketCounter%100 (from given ex. = "PC")
```

```
private static String getPacketPC(String ARQPacket) {
```

```
    return ARQPacket.split(" ")[3];
```

```
}
```

```
private static int getPacketPC_INT(String ARQPacket) {
```

```
    return Integer.valueOf(getPacketPC(ARQPacket));
```

```
}
```

```
// Returns the pseudoencrypted 16-char sequence (from given ex. = "XXXXXXXXXXXXXXXX", without ">" & "<")
```

```
private static String getPacketEncryptedSequence(String ARQPacket) {
```

```
    return ARQPacket.split(" ")[4].substring(1, 17);
```

```
}
```

```
// Returns Frame Check Sequence (from given ex. = "FCS")
```

```
private static String getFCS(String ARQPacket) {
```

```
    return ARQPacket.split(" ")[5];
```

```
}
```

```
private static int getFCS_INT(String ARQPacket) {
```

```
    return Integer.valueOf(getFCS(ARQPacket));
```

```
}
```

```
// -----
```

```
}
```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμo: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[14]



Image.java

```
package applications;

import ithakimodem.*;

// -----
//          LIBRARIES
// -----

import java.io.File;
import java.io.FileOutputStream;
import java.io.ByteArrayOutputStream;

/**
 * @apiNote This class contains methods in order to receive/create a .jpeg/.jpg image frame, from
 * the videoCoder hosted at http://ithaki.eng.auth.gr/netlab/video.html (and provides live
 * feed from a road near the School Of Engineering of Aristotle University Of Thessaloniki),
 * through Ithaki's Virtual Modem.
 */
public class Image {

    /**
     * This method sends a request to the VirtualModem using the given
     * ImageRequestCode and then uses a ByteArrayOutputStream to save the image's
     * data (bytes) sent SERIALLY by the VirtualModem.
     *
     * @param VirtualModem      Ithaki's modem we send the request to
     * @param ImageRequestCode The request code in order to receive an image
     *
     * @return The image's binary file's data in a byte array, from start of image
     *         delimiter to end of image delimiter (including the delimiters 4 bytes)
     */
    public static byte[] receiveImageBytes(Modem VirtualModem, String ImageRequestCode) {
        ByteArrayOutputStream ImageDataBuffer = new ByteArrayOutputStream();
        byte[] ImageData = null;

        // Sending request to the virtual modem using the request code given
        boolean VirtualModemResponse = VirtualModem.write((ImageRequestCode + "\r").getBytes());
        if (VirtualModemResponse) {
            // Getting virtual modem's "answer" to the request
            int VirtualModemOutput;

            // We receive the image data from the modem until we find the image's file
            // breaking flag
            // Since the modem sends a .jpg/.jpeg image, we use JPG_BreakingFlagIsFound()
            // method to check if we need to stop receiving image data
            while (!JPG_BreakingFlagIsFound(ImageDataBuffer)) {
                VirtualModemOutput = VirtualModem.read();
                ImageDataBuffer.write((byte) VirtualModemOutput);
            }

            ImageData = ImageDataBuffer.toByteArray();
        }

        return ImageData;
    }

    /**
     * This method generates an image file at the PC's directory:
     * DirectoryAndFileNameAndExtension given the image's data ImageData
     *
     * @param ImageData          the image's binary file's data in a byte array
     * @param DirectoryAndFileNameAndExtension directory to save the image file in the PC/filename.extension
     *
     * @return TRUE if image generated successfully,
     */
}
```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[15]

```
*          FALSE if an error occurred or the image is not generated successfully
*/
public static boolean generateImage(byte[] ImageData, String DirectoryAndFileNameAndExtension) {
    // Check if ImageData was generated SUCCESSFULLY
    if (!(ImageData == null)) {
        // Create the file at the location given by the user
        File ImageFile = new File(DirectoryAndFileNameAndExtension);

        // Use a FileOutputStream to write the image data to the image file
        try (FileOutputStream ImageStream = new FileOutputStream(ImageFile)) {
            ImageStream.write(ImageData);
        } catch (Exception x) {
            // Console output:
            System.out.println("Caught EXCEPTION: " + x + " ! Couldn't generate " + DirectoryAndFileNameAndExtension);
        }

        // Console output:
        // (CONFIRMATION MESSAGE)
        System.out.println();
        System.out.println();
        System.out.println("[IMAGE][ SUCCESSFULLY generated image: " + DirectoryAndFileNameAndExtension + " ]");
        System.out.println();
        System.out.println();

        return true;
    } else {
        // Console output:
        System.out.println("[IMAGE][ Couldn't generate .jpg image! No file data? ]");

        return false;
    }
}

/**
 * This method scans the Image's data ByteArrayOutputStream to find it's data
 * breaking flag Breaking flags are used to determine an image's type.
 * In a JPEG/JPG Image: The binary file STARTS with the bytes 0xFF 0xD8 (start of image delimiter)
 *                       & ENDS with the bytes 0xFF 0xD9 (end of image delimiter).
 *
 * @param ImageDataBuffer Image's current data ByteArrayOutputStream
 *
 * @return TRUE if JPG/JPEG's breaking flag is found at the end of the ImageDataBuffer,
 *         FALSE else
 */
private static boolean JPG_BreakingFlagIsFound(ByteArrayOutputStream ImageDataBuffer) {
    // We check if the ImageDataBuffer.toByteArray() has data and if it has more
    // than 4 bytes, since 4 bytes are used for the start of image delimiter
    // and the end of image delimiter
    if (ImageDataBuffer.toByteArray().length > 4) {

        // We check the second to last byte of the array for the byte: 0xFF
        // We check the last byte of the array for the byte: 0xD9
        if ((ImageDataBuffer.toByteArray()[ImageDataBuffer.toByteArray().length - 2] == ((byte) (0xFF)))
            && (ImageDataBuffer.toByteArray()[ImageDataBuffer.toByteArray().length - 1] == ((byte) (0xD9)))) {
            return true;    // The end of image delimiter is found at the end of the buffer!!
        } else
            return false;    // The end of image delimiter is NOT found at the end of the buffer
    } else
        return false; // Not enough bytes in the buffer to determine if image is sent
}
}
```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμo: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[16]



GPS.java

```
package applications;

import ithakimodem.*;

// -----
//          LIBRARIES
// -----
import java.util.ArrayList;
import java.lang.Math;

/**
 * @apiNote This class contains methods in order to receive and analyse Global
 * Positioning System (GPS) packets, following the NMEA serial Protocol.
 * This protocol anticipates a number of GPS sentences that carry position,
 * speed, time, satellite identity etc. in the form of ASCII characters.
 */
public class GPS {

    /**
     * This method sends a request to Ithaki's VirtualModem using the GPSRequestCode
     * in order to receive GPS (NMEA Protocol) packets and stores the latitude and
     * longitude of those packets in lists. Then analyses those lists in order to
     * find 4 map points with distance of at least 4'' between the each other.
     * At last, it creates a "T=" parameter for those map points and returns it
     *
     * @param VirtualModem      Ithaki's modem we send the request to
     * @param GPSRequestCode    The request code in order to receive GPS packets
     *
     * @return The "T=" parameter (FULL_TParameter) for the chosen 4 map points
     */
    public static String GPSApplication(Modem VirtualModem, String GPSRequestCode) {
        // Sending request to the virtual modem.
        boolean VirtualModemResponse = VirtualModem.write((GPSRequestCode + "\r").getBytes());
        // Getting virtual modem's "answer" to the request.
        if (VirtualModemResponse) {
            String VirtualModemOutput = "";
            String NMEAProtocol_GPS_FullString = "";

            // GPS Points LISTS (1 Holding it's Latitude, the other it's Longitude)(SAME INDEX)
            ArrayList<String> AllLatitudeCoordinates = new ArrayList<String>();
            ArrayList<String> AllLongitudeCoordinates = new ArrayList<String>();

            // Before sending the GPS Packets, Ithaki sends the message "START ITHAKI GPS TRACKING\r"
            // and at the end of the GPS packet stream, sends the message "STOP ITHAKI GPS TRACKING\r"
            //
            // We scan packets until we meet the end message.
            while (!VirtualModemOutput.endsWith("STOP ITHAKI GPS TRACKING\r")) {

                NMEAProtocol_GPS_FullString = getNextOutputLine(VirtualModem);          // Read next line (Empty or Packet)

                VirtualModemOutput += NMEAProtocol_GPS_FullString;                    // Save the line to our FULL OUTPUT String var.

                // ERROR HANDLING (In order to receive packets, Ithaki has to first send the "START ITHAKI GPS TRACKING\r")
                if ((VirtualModemOutput == "") || !(VirtualModemOutput.startsWith("START ITHAKI GPS TRACKING\r"))) {
                    // Console output:
                    System.out.println("Code ERROR or end of connection...");

                    return ""; // Return an EMPTY FULL_TParameter
                }

                // READ the lines starting with the GGA protocol header ("SGPGGA").
                // SKIP empty lines and then parse the GPS packet received
                if (NMEAProtocol_GPS_FullString.startsWith("$GPGGA")) {
                    // Save GPS Packet's Latitude and Longitude to our Points Lists
                    AllLatitudeCoordinates.add(getLatitude(NMEAProtocol_GPS_FullString));
                    AllLongitudeCoordinates.add(getLongitude(NMEAProtocol_GPS_FullString));

                    // Console output:
                    System.out.println("> Received GPS Packet (NMEA Protocol): " + NMEAProtocol_GPS_FullString);
                }
            }

            // DISTANCE CHOSEN GPS Points LISTS (1 Holding it's Latitude, the other it's Longitude)(SAME INDEX)
```


ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[17]

```
int NumberOfCoordsToSave = 5;
ArrayList<String> ChosenCoordsLIST_LATITUDE = new ArrayList<String>();
ArrayList<String> ChosenCoordsLIST_LONGITUDE = new ArrayList<String>();

String FULL_TParameter = "";

System.out.println("(Analysing GPS packets' points to find ones with a distance, between them, of at least 4''...)");
// Loop from Point #0 to Point #MAX and then find a second Point with distance greater or equal than 4'' to the first one
for (int i = 0; i < AllLatitudeCoordinates.size() && ChosenCoordsLIST_LATITUDE.size() < NumberOfCoordsToSave; i++) {
    for (int j = i; j < AllLatitudeCoordinates.size() && ChosenCoordsLIST_LATITUDE.size() < NumberOfCoordsToSave; j++) {
        // Check if the distance of Point #1 (i) and Point #2 (j) is equal or greater than 4''[ = DMS2DecRadians(0, 0, 4) ]
        if (distanceOfPointsRadians(
            // Point #1 Decimal-Degrees Coordinates
            // LATITUDE
            DDM2DecDegrees(getLatitudeDegreesINT(AllLatitudeCoordinates.get(i)),
                getLatitudeMinutesDOUBLE(AllLatitudeCoordinates.get(i))),
            // LONGITUDE
            DDM2DecDegrees(getLongitudeDegreesINT(AllLongitudeCoordinates.get(i)),
                getLongitudeMinutesDOUBLE(AllLongitudeCoordinates.get(i))),

            // Point #2 Decimal-Degrees Coordinates
            // LATITUDE
            DDM2DecDegrees(getLatitudeDegreesINT(AllLatitudeCoordinates.get(j)),
                getLatitudeMinutesDOUBLE(AllLatitudeCoordinates.get(j))),
            // LONGITUDE
            DDM2DecDegrees(getLongitudeDegreesINT(AllLongitudeCoordinates.get(j)),
                getLongitudeMinutesDOUBLE(AllLongitudeCoordinates.get(j)))
        ) >= DMS2DecRadians(0, 0, 4)) {
            // If distance check true:
            // Checks if the first point already exists in the "chosen" list due to a previous measurement
            if (!(ChosenCoordsLIST_LATITUDE.contains(AllLatitudeCoordinates.get(i))) &&
                (!ChosenCoordsLIST_LONGITUDE.contains(AllLongitudeCoordinates.get(i)))) {
                // If unique point true:
                // Save Point #1 Latitude and Longitude
                ChosenCoordsLIST_LATITUDE.add(AllLatitudeCoordinates.get(i));
                ChosenCoordsLIST_LONGITUDE.add(AllLongitudeCoordinates.get(i));
                System.out.println("[GPS] Point found: LATITUDE: " + AllLatitudeCoordinates.get(i)
                    + " , LONGITUDE: " + AllLongitudeCoordinates.get(i));
            }
            // Save Point #2 Latitude and Longitude ( Does not need to be checked, it can't already be in the list)
            ChosenCoordsLIST_LATITUDE.add(AllLatitudeCoordinates.get(j));
            ChosenCoordsLIST_LONGITUDE.add(AllLongitudeCoordinates.get(j));
            System.out.println("[GPS] Point found: LATITUDE: " + AllLatitudeCoordinates.get(j)
                + " , LONGITUDE: " + AllLongitudeCoordinates.get(j));

            i = j - 1; // If we find a valid distance between Point i and Point j we continue
                    // searching more Points after Point j ( j - 1 because loop does j++ )
            break;
        }
    }
}

// Generate the T= parameter for each point we have chosen and put them together
// ex "T=123456789123T=456789012345"
// where T=123456789123 : generateT_Parameter() for Point #1
// T=456789012345 : generateT_Parameter() for Point #2
// ...
// (Ithaki can get a max of 9 "T=.." parameters after the gps request code so i < 9)
for (int i = 0; i < ChosenCoordsLIST_LATITUDE.size() && i < 9; i++) {
    FULL_TParameter += generateT_Parameter(ChosenCoordsLIST_LATITUDE.get(i), ChosenCoordsLIST_LONGITUDE.get(i));
}

// Console output:
System.out.println("[GPS][ Those points create the following T= parameter: " + FULL_TParameter + " ]");

return FULL_TParameter;

} else

return ""; // Return an EMPTY FULL_TParameter in case of no response from the Virtual Modem

}

/**
 * Reads the VirtualModem's output and return the next line it returns
 * (Reads until "\n" is returned from the modem)
 *
 * @param VirtualModem
 *
 * @return Next line from the VirtualModem's output
 */
public static String getNextOutputLine(Modem VirtualModem) {
    String SingleLineText = "";
```

[18]

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[19]

```
//      * @param Degrees
//      * @param DecMinutes Decimal Minutes
//      *
//      * @return Decimal Radians
//      */
private static double DDM2DecRadians(int Degrees, double DecMinutes) {
    return Math.toRadians((double) Degrees + (DecMinutes / 60));
}

/**
 * Uses the 'HAVERINE' formula to calculate the absolute angular distance
 * between two points - that is, the shortest distance over the earth's
 * surface - In RADIANS.
 *
 * This formula is for calculations on the basis of a spherical earth
 * (ignoring ellipsoidal effects) - which is accurate enough for most purposes...
 * [In fact, the earth is very slightly ellipsoidal; using a spherical model
 * gives errors typically up to 0.3%]
 *
 * @param latPoint1 The FIRST point's LATITUDE in a Decimal Degrees form
 * @param lonPoint1 The FIRST point's LONGITUDE in a Decimal Degrees form
 * @param latPoint2 The SECOND point's LATITUDE in a Decimal Degrees form
 * @param lonPoint2 The SECOND point's LONGITUDE in a Decimal Degrees form
 *
 * @return
 */
private static double distanceOfPointsRadians(double latPoint1, double lonPoint1, double latPoint2, double lonPoint2) {
    // Difference of latitudes and longitudes
    double dLat = Math.toRadians(latPoint2 - latPoint1);
    double dLon = Math.toRadians(lonPoint2 - lonPoint1);

    // Convert Latitudes from DEGREES to RADIANS
    latPoint1 = Math.toRadians(latPoint1);
    latPoint2 = Math.toRadians(latPoint2);

    // Apply the HAVERINE formula
    double a = Math.pow(Math.sin(dLat / 2), 2) + Math.pow(Math.sin(dLon / 2), 2) * Math.cos(latPoint1) * Math.cos(latPoint2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

    return Math.abs(c);
}

/**
 * Methods to parse the NMEA Protocol GPS packets sent by the VirtualModem.
 *
 * @bibliography SiRF NMEA Reference Manual: http://ithaki.eng.auth.gr/netlab/sirf-nmea-reference-manual.pdf
 *
 * @param NMEAProtocol_GPS_FullString : A NMEA GPS packet sent by the VirtuaModem. Example:
 *      "$GPGGA,045208.000,4037.6331,N,02257.5633,E,1,07,1.5,57.8,M,36.1,M,,0000*6D"
 */
// -----
// Returns HEADER (from given ex. = "$GPGGA")
private static String getHeader(String NMEAProtocol_GPS_FullString) {
    return (NMEAProtocol_GPS_FullString.split(",")[0]);
}

// Returns UTC-TIME (from given ex. = "045208.000")
private static String getUTCtime(String NMEAProtocol_GPS_FullString) {
    return (NMEAProtocol_GPS_FullString.split(",")[1]);
}

// Returns LATITUDE - DDM (Degrees Decimal Minutes) (from given ex. = "4037.6331")
private static String getLatitude(String NMEAProtocol_GPS_FullString) {
    return (NMEAProtocol_GPS_FullString.split(",")[2]);
}

// Returns Latitude DEGREES (from given ex. = "40")
private static String getLatitudeDeg(String NMEAProtocol_GPS_String_LATITUDE) {
    return NMEAProtocol_GPS_String_LATITUDE.substring(0, 2);
}

private static int getLatitudeDegreesINT(String NMEAProtocol_GPS_String_LATITUDE) {
    return Integer.valueOf(getLatitudeDeg(NMEAProtocol_GPS_String_LATITUDE));
}

// Returns Latitude Decimal MINUTES (from given ex. = "37.6331")
private static String getLatitudeMin(String NMEAProtocol_GPS_String_LATITUDE) {
    return NMEAProtocol_GPS_String_LATITUDE.substring(2, 9);
}

private static double getLatitudeMinutesDOUBLE(String NMEAProtocol_GPS_String_LATITUDE) {
```

ΔΙΚΤΥΑ ΥΠΟΛΟΓΙΣΤΩΝ 1

Όνομ/νυμο: Κυπαρίσσης Κυπαρίσσης

A.E.M.: 10346

[20]

```
        return Double.valueOf(getLatitudeMin(NMEAProtocol_GPS_String_LATITUDE));
    }

//    // Returns NORTH/SOUTH Indicator (from given ex. = "N")
//    private static String getNorthSouthIndicator(String NMEAProtocol_GPS_FullString) {
//        return (NMEAProtocol_GPS_FullString.split(",")[3]);
//    }

//    // Returns LONGITUDE (from given ex. = "02257.5633")
//    private static String getLongitude(String NMEAProtocol_GPS_FullString) {
//        return (NMEAProtocol_GPS_FullString.split(",")[4]);
//    }

//    // Returns Longitude DEGREES (from given ex. = "022")
//    private static String getLongitudeDeg(String NMEAProtocol_GPS_String_LONGITUDE) {
//        return NMEAProtocol_GPS_String_LONGITUDE.substring(0, 3);
//    }

//    private static int getLongitudeDegreesINT(String NMEAProtocol_GPS_String_LONGITUDE) {
//        return Integer.valueOf(getLongitudeDeg(NMEAProtocol_GPS_String_LONGITUDE));
//    }

//    // Returns Longitude Decimal MINUTES (from given ex. = "57.5633")
//    private static String getLongitudeMin(String NMEAProtocol_GPS_String_LONGITUDE) {
//        return NMEAProtocol_GPS_String_LONGITUDE.substring(3, 10);
//    }

//    private static double getLongitudeMinutesDOUBLE(String NMEAProtocol_GPS_String_LONGITUDE) {
//        return Double.valueOf(getLongitudeMin(NMEAProtocol_GPS_String_LONGITUDE));
//    }

//    // Returns EAST/WEST Indicator (from given ex. = "E")
//    private static String getEastWestIndicator(String NMEAProtocol_GPS_FullString) {
//        return (NMEAProtocol_GPS_FullString.split(",")[5]);
//    }

//    // Returns the position fix indicator (from given ex. = "1")
//    private static String getPositionFixIndicator(String NMEAProtocol_GPS_FullString) {
//        return (NMEAProtocol_GPS_FullString.split(",")[6]);
//    }

//    // Returns the number of active satellites (from given ex. = "07" )
//    private static String getActiveSatellites(String NMEAProtocol_GPS_FullString) {
//        return (NMEAProtocol_GPS_FullString.split(",")[7]);
//    }

//    // ...
//    private static String getHDOP(String NMEAProtocol_GPS_FullString) {
//        return (NMEAProtocol_GPS_FullString.split(",")[8]);
//    }

//    // ...
//    private static String getSeaLevel(String NMEAProtocol_GPS_FullString) {
//        return (NMEAProtocol_GPS_FullString.split(",")[9]);
//    }

//    // ...
//    private static String getSeaLevelUNIT(String NMEAProtocol_GPS_FullString) {
//        return (NMEAProtocol_GPS_FullString.split(",")[10]);
//    }

//    // ...
//    private static String getDiffRefStationID(String NMEAProtocol_GPS_FullString) {
//        return ((NMEAProtocol_GPS_FullString.split(",")[14]).split("\\\\*")[0]);
//    }

//    //...
//    private static String getChecksum(String NMEAProtocol_GPS_FullString) {
//        return "*" + ((NMEAProtocol_GPS_FullString.split(",")[14]).split("\\\\*")[1]);
//    }
//    -----
}
```