ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

Τμημα ΗΜΜΥ. Τομεας Ηλεκτρονικης

Δικτυα Υπολογιστων Ι

Κώδικας εφαρμογής

Συντάκτης :
Χρηστος Χουτουριδης
ΑΕΜ:8997
cchoutou@ece.auth.gr

Διδάσκον :
Δημητριος Μητρακος
mitrakos@eng.auth.gr

17 Απριλίου 2020

Περιεχομενα

1. Περιγραφη

Για την ανάπτυξη του κώδικα χρησιμοποιήσαμε το περιβάλλον eclipse και το openjdk στην έκδοση 11. Επειδή η εφαρμογή λειτουργεί ως command line tool, χρησιμοποιήσαμε ακόμα το commons-cli από apache commons ώς εργαλείο ανάγνωσης των ορισμάτων του χρήστη. Η υλοποίηση της εφαρμογής είναι πολύ απλή. Δεν χρησιμοποιούμε interfaces ή κληρονομικότητα. Αντίθετα έχουμε χωρίσει την εφαρμογή σε αυτόνομα αντικείμενα. Τα αντικείμενα Com και Log παρέχουν τις βασικές λειτουργίες για την επικοινωνία με τον server και την έξοδο του προγράμματος αντίστοιχα. Τα αντικείμενα αυτά χρησιμοποιούνται απλώς μέσα υπόλοιπα. Χρησιμοποιούμε δηλαδή την απλούστερη δυνατή τεχνική (build by composition).

Τον κώδικα καθώς και την παραγόμενη τεκμηρίωσή του μέσω JavaDoc μπορείτε να την βρείτε στο παραδοτέο αρχείο στους καταλόγους src και doc. Τέλος στο προσωπικό αποθετήριο του συντάκτη υπάρχει όλος ο κώδικας της εφαρμογής, της εργασίας και της επεξεργασίας των αποτελεσμάτων. Παραθέτουμε παρακάτω τον κώδικα της εφαρμογής σε μορφή !!κειμένου!!, όπως αυτό ζητήθηκε από την εκφώνηση της εργασίας. Το κάθε κεφάλαιο αντιστοιχεί σε ξεχωριστό αρχείο του κώδικα.

2. virtualModem.java

```
/**
 * @file VirtualModem.java
 * @brief
 *    Contain the Main class for the project VirtualModem
 *
 * Usage examples:
 * ==========================
 * java -jar ./VirtualModem.jar -v -l Exxxx-$(date +%F-%T).log -e Exxxx 600
 * java -jar ./VirtualModem.jar -v -l Qxxx-Rxxxx-$(date +%F-%T).log -a Qxxxx Rxxxx 600
 * java -jar ./VirtualModem.jar -v -l Pxxxx_600_80_8p-$(date +%F-%T).log -p Pxxxx 600 80 8 P1124-$(date
 * java -jar ./VirtualModem.jar -v -l Mxxxx-$(date +%F-%T).log -g Mxxxx Mxxxx-$(date +%F-%T) 2
 * java -jar ./VirtualModem.jar -v -l Gxxxx-$(date +%F-%T).log -g Gxxxx Gxxxx-$(date +%F-%T) 2
 *
 * @author Christos Choutouridis AEM:8997
 * @email  cchoutou@ece.auth.gr
 */
package net.hoo2.auth.vmodem;

/** @name imports */
/** @{ */
```

```
import org.apache.commons.cli.*; /** command line utility */
/** @} */


/**
 * @class VirtualModem
 *
 * @brief This is the main control class of the program.
 *
 * This class includes the main function which read the user input
 * and create the appropriate object for the requested sequence to
 * retrieve data.
 */
public class VirtualModem {

    /** @name Data */
    /** @{ */
        CommandLine       line;       /** Command line for the argument list */
        CommandLineParser parser;     /** Parser for the argument list */
        Options           options;    /** Option configuration container */
        HelpFormatter     formatter;  /** An extra helper for -h */
        Com               com;        /** Reference to basic configuration module */
        Log               log;        /** Reference to basic logging module */
        /** @} */

    /**
     * @brief The main constructor of the class
     * We allocate memory for the data needed for the command line
     */
    public VirtualModem () {
        parser      = new DefaultParser();
        options     = new Options();
        formatter   = new HelpFormatter();
        com         = new Com();

        // Create argument options and a basic help file
        // for future use.
        Option verb    = new Option ("v", "verbose", false, "Be more verbose to the console");
        Option help    = new Option ("h", "help", false, "Print this message");
        Option timeout = Option.builder("t")
                         .longOpt("timeout")
                         .hasArg()
                         .valueSeparator('=')
                         .desc("Select timeout in [sec]")
                         .build();
        Option speed   = Option.builder("s")
                         .longOpt("speed")
                         .hasArg()
                         .valueSeparator('=')
                         .desc("Select speed in [bps]")
                         .build();
        Option log     = Option.builder("l")
                         .longOpt("log")
                         .hasArg()
                         .desc("Log file name to use")
                         .build();
        Option  echo   = Option.builder("e")
                         .longOpt("echo")
                         .numberOfArgs(2)
```

```
                    .desc ("Request echo sequence where <arg> = <1> <2>\n"
                    + "   <1>: The echo requested code from virtual lab\n"
                    + "   <2>: The time duration of the session [sec]")
                    .build();
    Option  arq    = Option.builder("a")
                    .longOpt("arq")
                    .numberOfArgs(3)
                    .desc ("Request arq sequence where <arg> = <1> <2> <3>\n"
                    + "   <1>: The ACK requested code for virtual lab\n"
                    + "   <2>: The NACK requested code from virtual lab\n"
                    + "   <2>: The time duration for the session [sec]")
                    .build();
    Option img     = Option.builder("g")
                    .longOpt("img")
                    .numberOfArgs(3)
                    .desc("Request an image sequence where <arg> = <1> <2> <3>\n"
                    + "   <1>: The image requested code from virtual lab\n"
                    + "   <2>: The filename to use for storing the image (without .jpg)\n"
                    + "   <3>: The requested images to fetch.")
                    .build();
    Option gps     = Option.builder("p")
                    .longOpt("gps")
                    .numberOfArgs(5) //  G8164 10 800 8 gps_10_800
                    .desc("Request a GPS sequence where <arg> = <1> <2> <3> <4>\n"
                    + "   <1>: The gps requested code from virtual lab\n"
                    + "   <2>: The time from trace to use as starting point [sec]\n"
                    + "   <3>: The time duration for the trace to fetch [sec]\n"
                    + "   <4>: The number of points to fetch from the above trace\n"
                    + "   <5>: The filename to use for storing the image (without .jpg)")
                    .build();
    options.addOption(verb);
    options.addOption(help);
    options.addOption(timeout);
    options.addOption(speed);
    options.addOption(log);
    options.addOption(echo);
    options.addOption(arq);
    options.addOption(img);
    options.addOption(gps);
}

/** @name private api */
/**@{*/

    /**
     * parse the command line arguments
     * @param args the arguments to parse
     * @return the status of the operation
     */
    private boolean getCmdOptions (String[] args) {
        try {
            // parse the command line arguments
            line = parser.parse (options, args);
        }
        catch( ParseException exp ) {
            // oops, something went wrong
            System.err.println( "Parsing command line failed: " + exp.getMessage() );
            return false;
        }
```

```
    return true;
}

/**
 * Dispatch the correct functionality based on arguments
 * @return the status of the operation (currently true)
 */
private boolean commandDispatcher () {
    boolean verbose = false;

    if (line.hasOption("help")) {
        formatter.printHelp( "virtualModem", options );
        return true;
    }

    // Get log and verbose options first
    if (line.hasOption("verbose"))
        verbose = true;
    if (line.hasOption("log"))
        log = new Log (line.getOptionValue("log"), verbose);
    else
        log = new Log (null, verbose);
    if (log.open() != true)
        return false;

    // get other options
    if (line.hasOption("timeout")) {
        com.timeout(Integer.parseInt(line.getOptionValue("timeout")));
    }
    if (line.hasOption("speed")) {
        com.speed(Integer.parseInt(line.getOptionValue("speed")));
    }

    // Execution dispatcher
    if (line.hasOption("echo")) {
        byte[] code = line.getOptionValues("echo")[0].getBytes();
        Echo e = new Echo(com, log, code,
                          Integer.valueOf(line.getOptionValues("echo")[1]));
        if (com.open()) {
            e.caption(code);
            e.run();
            com.close();
        }
    }
    else if (line.hasOption("arq")) {
        byte[]  ack = line.getOptionValues("arq")[0].getBytes();
        byte[] nack = line.getOptionValues("arq")[1].getBytes();
        ARQ a = new ARQ(com, log, ack, nack,
                        Integer.valueOf(line.getOptionValues("arq")[2]));
        if (com.open()) {
            a.caption(ack, nack);
            a.run();
            com.close();
        }
    }
    else if (line.hasOption("img")) {
        byte[] code = line.getOptionValues("img")[0].getBytes();
        Image im = new Image (com, log, code,
                              line.getOptionValues("img")[1],
```

```
                                    Integer.valueOf(line.getOptionValues("img")[2]));
                if (com.open()) {
                    im.caption(code);
                    im.run();
                    com.close();
                }
            }
            else if (line.hasOption("gps")) {
                byte[] code = line.getOptionValues("gps")[0].getBytes();
                GPS g = new GPS (
                            com, log, code,
                            Integer.valueOf(line.getOptionValues("gps")[1]),
                            Integer.valueOf(line.getOptionValues("gps")[2]),
                            Integer.valueOf(line.getOptionValues("gps")[3])
                        );
                Image im = new Image (
                                    com, log, null,
                                    line.getOptionValues("gps")[4], 1
                                );
                if (com.open()) {
                    g.caption();
                    im.code(g.run().getBytes());
                    im.run();
                    com.close();
                }
            }

            log.close();
            return true;
        }
        /**@}*/


    /**
     * @brief Main function
     */
    public static void main(String[] args) {
        // allocate the main object
        VirtualModem vmodem = new VirtualModem();

        // prepare command line input
        if (vmodem.getCmdOptions (args) != true)
            return;
        // Call the requested functionality
        if (vmodem.commandDispatcher() != true)
            return;
    }
}
```

3. Com.java

```
/**
 * @file Com.java
 *
 * @author Christos Choutouridis AEM:8997
 * @email  cchoutou@ece.auth.gr
 */
package net.hoo2.auth.vmodem;
```

```
/** @name imports */
/** @{ */
import java.util.Arrays;
import ithakimodem.*;
/** @} */

/**
 * @class Com
 *
 * Wrapper object of the provided Modem class to provide
 * a convenient and common API for all the session functionalities
 */
class Com {
    static final int SPEED_DEFAULT   = 48000;    /** default communication speed [bps] */
    static final int TIMEOUT_DEFAULT = 2000;     /** default timeout values [sec] */
    static final String URL_DEFAULT  = "ithaki"; /** Default destination. Ithaki of-course */

    private Modem  modem_;     /** Ref to Modem */
    private int    speed_;     /** Communication speed [bps] */
    private int    timeout_;   /** communication timeout [sec] */

    /**
    * Basic constructor
    */
    Com () {
        modem_   = new Modem();
        speed_   = SPEED_DEFAULT;
        timeout_ = TIMEOUT_DEFAULT;
        modem_.setSpeed(speed_);
        modem_.setTimeout(timeout_);
    }
    /** A more fancy constructor
     *
     * @param log     Reference to Log to use
     * @param speed   The desired speed [bps]
     * @param timeout The desired timeout [sec]
     * @note
     *    Currently not used
     */
    Com (Log log, int speed, int timeout) {
        modem_   = new Modem();
        speed_   = speed;
        timeout_ = timeout;
        modem_.setSpeed(speed_);
        modem_.setTimeout(timeout_);
    }

    /** @name Mutator interface */
    /**@{ */
        /** Get the current timeout value [sec] */
        int  timeout ()      { return timeout_; }
        /** Set the timeout value [sec] */
        void timeout (int timeout) {
            if (timeout_ != timeout) {
                timeout_ = timeout;
                modem_.setTimeout(timeout_);
            }
        }
        /** Get the current speed [bps] */
```

```
        int  speed () { return speed_;  }
        /** Set the speed [bps] */
        void speed (int speed) {
            if (speed_ != speed) {
                speed_ = speed;
                modem_.setSpeed(speed_);
            }
        }
        /**@} */

    /** @name Public API of the class */
    /**@{ */
        /** Open a communication channel with a URL */
        boolean open (String url) { return modem_.open(url); }
        /** Open a communication channel with Ithaki */
        boolean open ()           { return modem_.open(URL_DEFAULT); }
        /** Close the connection */
        boolean close()           { return modem_.close(); }

        /**
         * @grief The basic communication functionality
         *
         * This function sends a request to server and return the response. In order to
         * do that, the main reading loop seeks for the starting pattern and use the data
         * from that point up to the ending pattern. This is very helpful for binary incoming
         * data.
         * @note
         *    Blocking mode
         * @param data    Reference to Transaction data to use @see Transaction
         * @param code    The request code to send
         * @param begin   The starting pattern of the response data
         * @param end     The ending pattern of the response data
         * @param bin     A flag to indicate that the data are in binary form
         * @return        Reference to Transaction object with the response data. @see Transaction
         */
        Transaction request (Transaction data, byte[] code, byte[] begin, byte[] end, boolean bin) {
            int ch =0;
            boolean have_begin = (begin != null) ? false : true;
            boolean incoming   = false;

            // Clear the receive buffer first
            _clear (data.response, 0, data.response.length);
            if (code != null) {
                // if we have a request
                data.code = code;          // store the request code for the transaction
                modem_.write(data.code);   // send the code and mark the time
                modem_.write((int)'\r');
                data.departure = System.currentTimeMillis();// - (long)((8*(data.code.length+1))*(1000.
            }

            // main receive loop
            data.size =0;
            do {
                // escape guard for memory protection
                if (data.size >= data.response.length) {
                    data.size =0;
                    return data;
                }
                // read the data byte from server
```

```
            try {
                ch = modem_.read();
            }
            catch (Exception e) {
                // Oops!!!! something went wrong. Break and return :(
                System.err.println (e.getMessage());
                data.size =0;
                return data;
            }
            if (!incoming) {
                // first byte triggered. mark the time
                incoming = true;
                data.arrival = System.currentTimeMillis();// - (long)(8*(1000.0/speed_));
            }
            data.response [data.size++] = (byte)ch;   // store the byte

            if (!have_begin && (detect(data.response, begin, data.size) != -1)) {
                // If begin pattern detected, start over.
                _clear(data.response, 0, data.size);
                data.size = _copy (data.response, begin);
                have_begin = true;
            }
        } while (
            ch != -1 && (
                !have_begin || (
                    (bin || (detect (data.response, "\r\n\n\n".getBytes(), data.size) == -1))
                    && (bin || (detect (data.response, "NO CARRIER".getBytes(), data.size) == -1))
                    && (detect (data.response, end, data.size) == -1)
                )
            )
        );
        /*^
         * We loop until:
         *    1) server returns -1
         *    2) A end pattern arrives
         *    3) A "\r\n\n\n" sequence arrives (in character mode, not for binary files)
         *    4) A "NO CARRIER" sequence arrives (in character mode, not for binary files)
         */
        return data;
    }

    /**
     * @brief
     *    A pattern detect functionality.
     *
     * Search in data input for the pattern and return its position if found.
     * @note
     *    O(n^2) in time.
     * @param data       The data buffer to search into
     * @param pattern    The pattern to search for
     * @param max        The maximum length to search
     * @return           The position of pattern in data, or -1 if not found
     */
    static int detect (byte[] data, byte[] pattern, int max) {
        if (pattern != null) {
            for (int i =0 ; i<max && i<data.length - pattern.length; ++i) {
                boolean detected = true;
                for (int j=0 ; j<pattern.length ; ++j) {
                    if (data[i+j] != pattern[j]) {
```

```
                            detected = false;
                            break;
                        }
                    }
                    if (detected)
                    return i;
                }
            }
            return -1;
        }
    /**@} */

    /** @name Private helper api */
    /**@{*/
        /**
         * Clear a buffer segment
         * @param buffer     Buffer to clear
         * @param begin      The starting point
         * @param end        The ending point
         */
        private void _clear (byte[] buffer, int begin, int end) {
            for (int i=begin ; i<end && i<buffer.length ; ++i)
            buffer[i] = 0;
        }

        /**
         * Copy a buffer to another
         * @param dest    The destination buffer
         * @param src     The source buffer
         * @return        The number of bytes copied
         */
        private int _copy (byte[] dest, byte[] src) {
            if (dest.length >= src.length) {
                for (int i=0 ; i<src.length ; ++i)
                dest[i] = src[i];
                return src.length;
            }
            return 0;
        }
    /**@}*/
}

/**
* @class
*    Class to represent client-server transaction data
*/
class Transaction {
    byte[]   code;       /** request code from client */
    byte[]   response;   /** response data from server */
    long     departure;  /** request time */
    long     arrival;    /** response time */
    int      size;       /** size of response data in bytes */

    /**
     * Basic constructor
     * @param code       Code to use
     * @param response   Response buffer to use
     */
    Transaction (byte[] code, byte[] response) {
```

```
        this.code      = code;
        this.response  = response;
        departure      = arrival = 0;
        size           = 0;
    }

    /**
     * Get a copy of the response
     */
    byte[] getResponse() {
        return Arrays.copyOf(response, size);
    }
}
```

4. Log.java

```
/**
 * @file Log.java
 *
 * @author Christos Choutouridis AEM:8997
 * @email  cchoutou@ece.auth.gr
 */
package net.hoo2.auth.vmodem;

/** @name imports */
/** @{ */
import java.io.IOException;
import java.io.PrintWriter;
/** @} */

/**
 * @class Log
 *
 * A common log functionality class for all sessions
 */
class Log {

    private String      logfile_;    /** The log file name */
    private boolean     verbose_;    /** The desired verbosity (for the console)*/
    private PrintWriter writer_;     /** A buffered writer to use for streaming */

    /**
     * Basic constructor
     * @param logfile    The log filename
     * @param verbose    The desired verbosity (for the console)
     */
    Log (String logfile, boolean verbose) {
        logfile_ = logfile;
        verbose_ = verbose;
    }

    /**
     * Try to open the log file
     * @return  The status of the operation
     */
    boolean open () {
        if (logfile_ != null) {
            try {
                writer_ = new PrintWriter(logfile_);
```

```
            }
            catch (IOException exp) {
                System.err.println( "Open log file failed: " + exp.getMessage() );
                return false;
            }
        }
        return true;
    }
    /**
     * Try to open a log file
     * @param logfile    The log file to open
     * @return           The status of the operation
     */
    boolean open (String logfile) {
        logfile_ = logfile;
        return open();
    }

    /**
     * Close the opened file
     * @return     The status of the operation
     */
    boolean close () {
        try {
            if (writer_ != null)
            writer_.close();
        } catch (Exception ex) {
            return false;
        }
        return true;
    }

    /**
     * Log to file and print to console
     * @param line    The line to log
     * @param out     Console output request flag. If true, echo the line to console
     */
    void write (String line, boolean out) {
        if (logfile_ != null) writer_.println(line);
        if (verbose_ || out)  System.out.println(line);
    }

    /**
     * Log to file and print to console
     * @param line    The line to log
     */
    void write (String line) {
        if (logfile_ != null) writer_.println(line);
        if (verbose_)         System.out.println(line);
    }

    /**
     * Echo the line to console
     * @param line    The line to print
     */
    void out (String line) {
        if (verbose_)  System.out.println(line);
    }
}
```

5. ARQ.java

```
/**
 * @file ARQ.java
 *
 * @author Christos Choutouridis AEM:8997
 * @email  cchoutou@ece.auth.gr
 */
package net.hoo2.auth.vmodem;

/** @name imports */
/** @{ */
import java.util.*;
/** @} */

/**
 * @class ARQ
 *
 * Class to used for the ACK-NACK sequence
 */
class ARQ {
    static final int ARQ_DURATION_DEFAULT  = 240;      /** Default duration for the sequence */
    static final int ARQ_BUFFER_SIZE       = 256;      /** ARQ buffer size */
    static final String ARQ_BEGIN          = "PSTART"; /** Begin pattern of the response */
    static final String ARQ_END            = "PSTOP";  /** End pattern of the response */

    static final int ARQ_SEQUENCE_BEGIN    = 31;    /** The response sequence string position */
    static final int ARQ_SEQUENCE_END      = 47;    /** The end of response sequence string position */
    static final int ARQ_CRC_BEGIN         = 49;    /** The response crc string position */
    static final int ARQ_CRC_END           = 52;    /** The end of response crc string position */

    private Com          com_;             /** Reference to communication module */
    private Log          log_;             /** Reference to logging module */
    private Transaction  transaction_;     /** A transaction object to used as a buffer for all request
    private int          duration_;        /** The desired duration for the session */
    private byte[]       ack_;             /** The desired ACK code for the session */
    private byte[]       nack_;            /** The desired NACK code for the session */

    /**
     * Basic constructor
     * @param com        The Com module to use
     * @param log        The Log module to use
     * @param ack        The desired ACK code
     * @param nack       The desired NACK code
     * @param duration   The desired duration for the session
     */
    ARQ (Com com, Log log, byte[] ack, byte[] nack, int duration) {
        com_        = com;
        log_        = log;
        duration_   = duration;
        transaction_= new Transaction(null, new byte[ARQ_BUFFER_SIZE]);
        ack_        = ack;
        nack_       = nack;
    }

    /**
     * Functionality to drain the response buffer for the welcome message
     * from the server
     * @param ack     The ack code that we will use (for printing, not sending)
     * @param nack    The nack code that we will use (for printing, not sending)
```

```
 */
void caption (byte[] ack, byte[] nack) {
    String line;

    line = "Running ARQ with: " + new String(ack) + "/" + new String(nack);
    log_.write(line, true);

    transaction_ = com_.request (transaction_, null, null, null, false);
    line = new String(transaction_.getResponse());
    log_.out(line);
}

/**
 * Main transaction loop. It send requests to server, get the response
 * and log the procedure while doing it.
 */
void run () {
    long     start;
    long     now;
    long     mark =0, tr =0, ts =0, tt =0;
    String   line;
    int      errors = 0;
    boolean  good = true;

    start = System.currentTimeMillis();
    do {
        // Check if the previous transaction was error-free
        if (good)
            transaction_ = com_.request(transaction_, ack_, ARQ_BEGIN.getBytes(), ARQ_END.getBytes(
        else
            transaction_ = com_.request(transaction_, nack_, ARQ_BEGIN.getBytes(), ARQ_END.getBytes
        good = (_crc_check(transaction_.response)) ? true : false;  // crc check the response

        // time calculations
        tr = transaction_.arrival - transaction_.departure;
        tt = 0;
        if ((errors == 0) && good) {
            tt = ts = tr;
        }
        if ((errors == 0) && !good) {
            mark = transaction_.departure;
            ts = tr;
        }
        if ((errors != 0) && good) {
            tt = transaction_.arrival - mark;
            ts += tr;
        }
        if ((errors !=0) && !good) {
            ts += tr;
        }
        errors = (good) ? 0 : errors+1;   // update the error strike

        line = new String(transaction_.code) + ": "
                    + new String(transaction_.getResponse())
                    + " Er: " + errors
                    + " Tr= " + tr + " [msec]";
        if (errors !=0)
            line += " Ts= 0 [msec] Tt= 0 [msec]";
        else
```

```
                line += " Ts= " + ts + " [msec]" + " Tt= " + tt + " [msec]";

            log_.write(line);
            now = System.currentTimeMillis();
        } while (now - start < duration_*1000);
    }

    /** @name private helper API */
    /**@{ */

    /**
     * A CRC check functionality
     * @param data    The data to check for validity
     * @return        The check result
     */
    private boolean _crc_check (byte[] data) {
        byte[] seq = Arrays.copyOfRange(data, ARQ_SEQUENCE_BEGIN, ARQ_SEQUENCE_END);
        int crc    = Integer.valueOf(
                        new String(Arrays.copyOfRange(data, ARQ_CRC_BEGIN, ARQ_CRC_END))
                    );
        return (crc == _crc(seq)) ? true : false;
    }

    /**
     * A CRC calculation function. This primitive CRC calculator
     * does not use any known CRC polynomial. It just uses XOR
     * @param data    Reference to buffer
     * @return        The CRC result
     */
    private int _crc (byte[] data) {
        int calc =0;
        for (int i=0 ; i<data.length ; ++i)
        calc ^= data[i];
        return calc;
    }
    /**@} */
}
```

6. Echo.java

```
/**
 * @file Echo.java
 *
 * @author Christos Choutouridis AEM:8997
 * @email  cchoutou@ece.auth.gr
 */
package net.hoo2.auth.vmodem;

/**
 * @class Echo
 *
 * Class to used for the echo sequence
 */
class Echo {
    static final int ECHO_DURATION_DEFAULT = 240;      /** Default duration for the sequence */
    static final int ECHO_BUFFER_SIZE      = 256;      /** Echo buffer size */
    static final String ECHO_BEGIN         = "PSTART"; /** Begin pattern of the response */
    static final String ECHO_END           = "PSTOP";  /** End pattern of the response */
```

```
    private Com          com_;             /** Reference to communication module */
    private Log          log_;             /** Reference to logging module */
    private Transaction  transaction_;     /** A transaction object to used as a buffer for all request
    private int          duration_;        /** The desired duration for the session */
    private byte[]       code_;            /** The desired code for the session */

    /**
     * Basic constructor
     * @param com        The Communication module to use
     * @param log        The logging module to use
     * @param code       The code to use
     * @param duration   The duration to use
     */
    Echo (Com com, Log log, byte[] code, int duration) {
        com_        = com;
        log_        = log;
        duration_   = duration;
        // Allocate memory for the response
        transaction_= new Transaction(null, new byte[ECHO_BUFFER_SIZE]);
        code_       = code;
    }

    /**
     * Functionality to drain the response buffer for the welcome message
     * from the server
     * @param code    The code that we will use (for printing, not sending)
     */
    void caption (byte[] code) {
        String line;

        line = "Running ECHO with: " + new String(code);
        log_.write(line, true);

        transaction_ = com_.request (transaction_, null, null, null, false);
        line = new String(transaction_.getResponse());
        log_.out(line);
    }

    /**
     * Main transaction loop. It send requests to server, get the response
     * and log the procedure while doing it.
     */
    void run () {
        long start;
        long now;
        String line;

        start = System.currentTimeMillis();
        do {
            transaction_ = com_.request(transaction_, code_, ECHO_BEGIN.getBytes(), ECHO_END.getBytes()
            line = new String(transaction_.code) + ": "
                        + new String(transaction_.getResponse())
                        + " Tr= " + (transaction_.arrival - transaction_.departure) + " [msec]";
            log_.write(line);
            now = System.currentTimeMillis();
        } while (now - start < duration_*1000);
    }

}
```

7. GPS.java

```
/**
 * @file GPS.java
 *
 * @author Christos Choutouridis AEM:8997
 * @email  cchoutou@ece.auth.gr
 */
package net.hoo2.auth.vmodem;

/** @name imports */
/** @{ */
import java.util.Arrays;
/** @} */

/**
 * @class GPS
 *
 * Class to used for the GPS session
*/
class GPS {
    static final int    GPS_BUFFER_SIZE    = 256;   /** GPS trace buffer size */
    static final String GPS_BEGIN          = "START ITHAKI GPS TRACKING";   /** starting pattern */
    static final String GPS_END            = "STOP ITHAKI GPS TRACKING";    /** ending pattern */
    static final int    GPS_USE_TRACK      = 1;     /** Which track to use (given) */
    static final String GPS_TRACK_PREFIX   = "R=";  /** GPS command track request prefix */
    static final String GPS_IMAGE_PREFIX   = "T=";  /** GPS command image request prefix */
    static final int    GPS_MAX_POINTS     = 9;     /** Maximum points (given) */
    static final int    GPS_COORDINATES_SIZE = 6;   /** Coordinates size */

    static final int    GPS_LATITUDE_BEGIN = 17;    /** The latitude sequence string position */
    static final int    GPS_LATITUDE_END   = 28;    /** The end of latitude sequence string position */
    static final int    GPS_LONGITUDE_BEGIN= 29;    /** The longitude sequence string position */
    static final int    GPS_LONGITUDE_END  = 41;    /** The end of longitude sequence string position *

    private Com          com_;          /** Reference to communication module */
    private Log          log_;          /** Reference to logging module */
    private Transaction  transaction_;  /** A transaction object to used as a buffer for all requests *
    private byte[]       code_;         /** The desired code for the session */
    private int          start_;        /** Starting point [sec] */
    private int          duration_;     /** Duration of requested trace */
    private int          points_;       /** Number of points to fetch from the above trace */
    private byte[]       coordinates_;  /** Coordinates buffer */

    /**
     * Basic constructor
     */
    GPS (Com com, Log log, byte[] code, int start, int duration, int points) {
        com_        = com;
        log_        = log;
        transaction_= new Transaction(null, new byte[GPS_BUFFER_SIZE]);
        code_       = code;
        start_      = start;
        duration_   = duration;
        points_     = (points <= GPS_MAX_POINTS) ? points : GPS_MAX_POINTS;
        coordinates_= new byte[GPS_COORDINATES_SIZE];
    }

    /**
     * Functionality to drain the response buffer for the welcome message
```

```
 * from the server
 */
void caption () {
    String line;

    line = "Running GPS with: " + new String(code_)
         + " time [" + start_ + " - " +(start_+duration_) + ") sec" + " [" + points_ + " points]";
    log_.write(line, true);

    transaction_ = com_.request (transaction_, null, null, null, false);
    line = new String(transaction_.getResponse());
    log_.out(line);
}

/**
 * Main transaction loop. It send requests to server, get the response
 * and log the procedure while doing it.
 * @return the image request code to pass to image module
*/
String run () {
    String code, image_code;
    String line;

    log_.out("Get traces");
    image_code = new String(code_);
    for (int trace =start_ ; trace < start_+duration_ ; trace += duration_/points_) {
        code = new String(code_) + GPS_TRACK_PREFIX + GPS_USE_TRACK + String.format("%04d", trace)
        transaction_ = com_.request(transaction_, code.getBytes(), GPS_BEGIN.getBytes(), GPS_END.ge
        line = new String(transaction_.code) + ": "
             + new String(transaction_.getResponse())
             + " Tr= " + (transaction_.arrival - transaction_.departure) + " [msec]";
        log_.write(line);

        _get_coordinates(transaction_.getResponse());
        image_code += GPS_IMAGE_PREFIX +
        String.format("%02d%02d%02d%02d%02d%02d", coordinates_[0], coordinates_[1],
        coordinates_[2], coordinates_[3],
        coordinates_[4], coordinates_[5]);

    }
    return image_code;
}

/** @name private helper API */
/** @{ */

/**
 * Extract coordinates from response
 * @param stream  The stream to search
 * @return        The coordinates buffer
 */
private byte[] _get_coordinates (byte[] stream) {
    int start        = Com.detect(stream, "GPGGA".getBytes(), stream.length);
    double latitude  = _nmea2dec (Double.valueOf(
    new String (Arrays.copyOfRange(stream,
    start+GPS_LATITUDE_BEGIN,
    start+GPS_LATITUDE_END-2))));
    double longitude = _nmea2dec (Double.valueOf(
    new String(Arrays.copyOfRange(stream,
```

```
        start+GPS_LONGITUDE_BEGIN,
        start+GPS_LONGITUDE_END-2))));

        coordinates_[0] = (byte)(longitude);                                  // longitude deg
        coordinates_[1] = (byte)((longitude - coordinates_[0]) * 60.0);   // longitude '
        coordinates_[2] = (byte)((longitude - coordinates_[0]
                                - coordinates_[1]/60.0)*3600);          // longitude "

        coordinates_[3] = (byte)(latitude);                                   // latitude deg
        coordinates_[4] = (byte)((latitude - coordinates_[3]) * 60.0);    // latitude '
        coordinates_[5] = (byte)((latitude - coordinates_[3]
                                - coordinates_[4]/60.0)*3600);          // latitude "
        return coordinates_;
    }

    /**
     * A helper to convert the NMEA format to canonical
     * decimal coordinate format
     */
    double _nmea2dec (double c) {
        int d = (int)c/100;
        c -= d*100;
        return d + (c/60);
    }
    /** @} */
}
```

8. Image.java

```
/**
 * @file Image.java
 *
 * @author Christos Choutouridis AEM:8997
 * @email  cchoutou@ece.auth.gr
 */
package net.hoo2.auth.vmodem;

/** @name imports */
/** @{ */
import java.io.*;
/** @} */

/**
 * @class Image
 *
 * Class to used for the error free and non error free image requests
 */
class Image {
    static final int     IMAGE_BUFFER_SIZE    = 256*1024;    /** image buffer size */
    static final byte[]  IMAGE_BEGIN          = {(byte)0xFF, (byte)0xD8};   /** jpeg image begin patter
    static final byte[]  IMAGE_END            = {(byte)0xFF, (byte)0xD9};   /** jpeg image end pattern

    private Com          com_;             /** Reference to communication module */
    private Log          log_;             /** Reference to logging module */
    private Transaction  transaction_;     /** A transaction object to used as a buffer for all request
    private String       filename_;        /** The filename to store */
    private int          items_;           /** how many images to fetch */
    private byte[]       code_;            /** The image request code for the virtual lab */
```

```
/**
 * Basic constructor
 * @param com        The com module to use
 * @param log        The log module to use
 * @param code       The request code
 * @param filename   The filename
 * @param items      How many items to fetch
 */
Image (Com com, Log log, byte[] code, String filename, int items) {
    com_        = com;
    log_        = log;
    // Allocate memory for the response
    transaction_= new Transaction(null, new byte[IMAGE_BUFFER_SIZE]);
    filename_   = filename;
    items_      = items;
    code_       = code;
}

/** Get function for the code */
void code (byte[] code) { code_ = code; }

/**
 * Functionality to drain the response buffer for the welcome message
 * from the server
 * @param code    The code that we will use (for printing, not sending)
 */
void caption (byte[] code) {
    String line;

    line = "Running video decoder with: " + new String(code);
    log_.write(line, true);

    transaction_ = com_.request (transaction_, null, null, null, false);
    line = new String(transaction_.getResponse());
    log_.out(line);
}

/**
 * Main transaction loop. It send requests to server, get the response
 * and log the procedure while doing it.
 */
void run () {
    String file, line;
    BufferedOutputStream ostream;

    for (int i =1 ; i<= items_ ; ++i) {
        // Make the filename string
        if (items_>1)
            file = filename_ + "_" + i + ".jpg";
        else
            file = filename_ + ".jpg";
        line = new String(code_) + ": ";
        log_.write(line);
        transaction_ = com_.request(transaction_, code_, IMAGE_BEGIN, IMAGE_END, true);
        line = "File= " + file
             + " Tr= " + (transaction_.arrival - transaction_.departure) + " [msec]"
             + " Tt= " + (System.currentTimeMillis() - transaction_.departure) + " [msec]";
        log_.write(line);
        try {
```

```
                ostream = new BufferedOutputStream(new FileOutputStream(file));
                ostream.write(transaction_.response);
                ostream.flush();
                ostream.close();
            }
            catch (Exception exp) {
                System.err.println ("Error creating " + file + exp.getMessage());
                return;
            }
        }
    }
}
```