אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

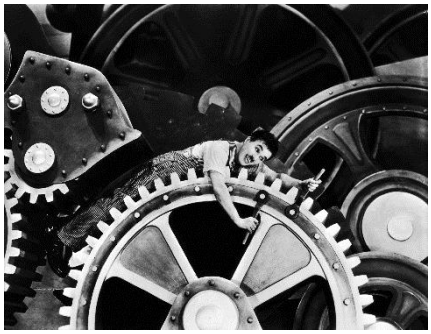
רשמו תשובותיכם בגיליון התשובות בלבד
תשובות מחוץ לגיליון לא יבדקו.

שימו לב:
על תשובות ריקות יינתן 20% מהניקוד!

בהצלחה!

תאריך הבחינה: 09.02.2015
שם המורה: ד"ר אנדרי שרף
ד"ר רן אטינגר
ד"ר ג'ון מרברג
ד"ר מני אדלר
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשע"ה
סמסטר: א'
מועד: א'
משך הבחינה: שלש שעות
חומר עזר: אסור

שאלה 1 (30 נקודות)



גלגלי שיניים הינם רכיבים אלמנטריים בכל מכונה מודרנית. ראשיתם במכונות שהמציא ארכימדס, אי שם במאה השלישית לפני הספירה. בצמד גלגלי שיניים, כאשר גלגל אחד מסתובב לכיוון מסוים, הגלגל השני מסתובב בכיוון ההפוך. מערכת גלגלי שיניים, הנקראת גם תמסורת, כוללת פעמים רבות גלגלי שיניים בגדלים שונים.
בשאלה זו נעסוק בסימולציה פשוטה של מערכת גלגלי שיניים.

הממשק Gear מגדיר גלגל שיניים ע"פ נקודות המרכז שלו (קואורדינטות x,y), אורך הרדיוס, ושני גלגלי השיניים שמצידו הימני והשמאלי (לפי הקואורדינטה x, כאשר קואורדינטת y אינה בהכרח זהה). כמו כן מגדיר הממשק את המתודה move, המסובבת את הגלגל בכיוון הנתון (עם או נגד כיוון השעון), ואת הגלגלים משני צדדיו בכיוון ההפוך.

```
interface Gear {
        float getX(); // returns the x coordinate of the center
        float getY(); // returns the y coordinate of the center
        float getRadius(); // returns the length of the radius
        Gear getLeft(); // returns the gear on the left
        Gear getRight(); // returns the gear on the right
        void move(boolean clockwise); // advance the gear and both its neighbors
        void moveLeft(Boolean clockwise); // advance the gear and its left neighbor
        void moveRight(Boolean clockwise); // advance the gear and its right neighbor
}
```

א. הגדירו תכונה נשמרת (invariant) עבור הממשק Gear (5 נקודות)
להלן מימוש של הממשק

```
class SimpleGear implements Gear  {
        final float _x,_y,_radius;
        Gear _left, _right;

        SimpleGear(float x, float y, float radius) throws Exception {
                if (!check(x,y,radius))   throw new Exception("Wrong parameter values!");
                _x = x; _y = y; _radius = radius; _left = null; _right = null;
        }

        public float getX() { return _x; }
        public float getY() { return _y; }
        public float getRadius() { return _radius; }
        public Gear getLeft() { return _left; }
        public Gear getRight() { return _right; }

        public void setLeft(Gear left) throws Exception {
                if (_left != null) throw new Exception("Left gear is already defined");
                if (!check(left. true)) throw new Exception("Wrong gear position!");
                _left = left;
        }

        public void setRight(Gear right) throws Exception {
                if (_right != null) throw new Exception("Right gear is already defined");
                if (!check(right,false)) throw new Exception("Wrong gear position!");
                _right = right;
        }

        public void move(boolean clockwise) {
                if (_left != null)
                        getLeft().moveLeft(!clockwise);
                if (_right != null)
                        getRight().moveRight(!clockwise);
        }

        public void moveLeft(boolean clockwise) {
                if (_left != null)
                        getLeft().moveLeft(!clockwise);
        }

        public void moveRight(boolean clockwise) {
                if (_right != null)
                        getRight().moveRight(!clockwise);
        }
```

```
        protected boolean check(float x, float y, float radius) {
                //@TODO
        }
        protected boolean check(Gear other, Boolean whichSide) {
                //whichSide indicates left-side (true) or right-side (false)
                //@TODO
        }
}
```

ב. השלימו את שתי המתודות check הבודקות את תקינות פרמטרי האתחול, ע"פ התכונה הנשמרת שציינתם בסעיף א (2 נקודות)

ג. בהנחה שמימוש המתודות check נכון, האם המחלקה בטוחה תחת כל חישוב מקבילי אפשרי? אם לא, הראו תרחיש מתאים. אם כן, נמקו בקצרה (4 נקודות)

למחלקה SimpleGear נוסף כעת שדה חדש _state המאפיין את מצב גלגל השיניים על ידי מספר ממשי המציין את הזווית שלו (כך שהערך הוא בין 0 ל 360). המצב ההתחלתי הוא 0. תזוזה בכיוון או נגד כיוון השעון מעלה או מורידה את הערך בהתאם, כפי שנוסף למתודות move, moveLeft, moveRight
[גודל התזוזה של גלגל השיניים תלוי ברדיוס שלו, כפי שמחושב ע"י המתודה הסטטית getOffset – אופן מימושה אינו רלבנטי כאן]

```
class SimpleGear implements Gear  {
        final float _x,_y,_radius;
        Gear _left, _right;
        float _state;

        SimpleGear(float x, float y, float radius) throws Exception {
                if (!check(x,y,radius))
                        throw new Exception("Wrong parameter values!");
                _x = x; _y = y; _radius = radius;
                _left = null; _right = null;
                _state = 0;
        }
        …
        public void move(boolean clockwise) {
                float offset = getOffset(_radius);
                if (clockwise)
                        _state = (_state + offset) % 360;
                else
                        _state = (_state - offset) % 360;
                if (_left != null)
                        getLeft().moveLeft(!clockwise);
                if (_right != null)
                        getRight().moveRight(!clockwise);
        }
```

```
        public void moveLeft(boolean clockwise) {
                float offset = getOffset(_radius);
                if (clockwise)
                        _state = (_state + offset) % 360;
                else
                        _state = (_state - offset) % 360;
                if (_left != null)
                        getLeft().moveLeft(!clockwise);
        }

        public void moveRight(boolean clockwise) {
                float offset = getOffset(_radius);
                if (clockwise)
                        _state = (_state + offset) % 360;
                else
                        _state = (_state - offset) % 360;
                if (_right != null)
                        getRight().moveRight(!clockwise);
        }

        protected static float getOffset(float radius) {
          // calculates the offset of a gear with a given radius
          // the implementation is not relevant for the question
        }
        ...
}
```

ד. עדכנו את התכונה הנשמרת. האם כעת המחלקה בטוחה תחת כל חישוב מקבילי אפשרי? אם לא, הראו תרחיש מתאים. אם כן, נמקו בקצרה (5 נקודות)

ה. הגדירו תנאי התחלה וסיום למתודה move (4 נקודות)

נתונה מערכת המסמלצת מנגנון פשוט של גלגלי שיניים:

```
class Spinner implements Runnable {
        Gear _gear;
        boolean _clockwise;

        Spinner(Gear gear, boolean clockwise) {
                _gear = gear;
                _clockwise = clockwise;
        }
```

```
        public void run() {
                for (int i=0;i<10; i++) {
                        _gear.move(_clockwise);
                        try {
                                Thread.sleep(1000);
                        } catch (InterruptedException e) {}
                }
        }
}

class Simulation {
        public static void main(String[] args) throws Exception {
                SimpleGear gear1 = new SimpleGear(10,10,5);
                SimpleGear gear2 = new SimpleGear(20,10,5);
                SimpleGear gear3 = new SimpleGear(30,10,5);
                gear2.setLeft(gear1);
                gear2.setRight(gear3);
                gear1.setRight(gear2);
                gear3.setLeft(gear2);
                new Thread(new Spinner(gear2,true)).start();
                new Thread(new Spinner(gear2,false)).start();
        }
}
```

ו. האם הרצת המערכת נכונה? (בבדיקת קריטריון הנכונות, כהתאמת סדרת המצבים האפשרית עבור הרצה בסדר כל שהוא של פעולות, התייחסו להפעלת המתודה **move** כפעולה). במידה ולא, תקנו את הקוד כך שיבטיח נכונות. (10 נקודות).

שאלה 2 (30 נקודות)

וקטור (vector) הינו מבנה אחסון אשר מחזיק מערך (array) באופן סדרתי בזיכרון. התכונה המיוחדת של וקטור היא שניתן להגדיל ולהקטין אותו (resize) באופן דינאמי על פי הצורך. לאחר שהתברר כי המחלקה **std::vector** בזבזנית ביחס להקצאות זיכרון, הוחלט לממש מחלקה חדשה בשם **mvector** באמצעות רשימה מקושרת, הדואגת שהזיכרון הנצרך על ידי הווקטור הוא בדיוק בגודל הנדרש על מנת לאחסן את הנתונים.

א. ממשו את פונקציות המחלקה **mvector**. ניתן להוסיף פונקציות עזר אך אין להוסיף שדות או לשנות חתימות של פונקציות קיימות. הגישה לאיבר ברשימה מקושרת אינה ב- O(1) כפי שבמערך. הפונקציה **insertAt** מכניסה איבר במיקום **pos**, תמיד מצליחה, ומקצה זיכרון אם נדרש. הפונקציה **removeAt** מוציאה איבר במיקום **pos**, ומקטינה את המבנה בהתאם. במידה ואין איבר במיקום זה, יוחזר איבר ריק כלשהו: **T dummy** (15 נקודות).

```
template <class T> class node {
public:
        node();
        node(const T &val);
        ~node();
```

```
private:
        const T&  _data;
        node *_next;
};

template <class T> class mvector {
public:
        mvector();
        ~mvector();
        mvector(const mvector &);
        void operator=(const mvector &);
        unsigned getSize();
        void insertAt(const T &, unsigned pos);
        const T & removeAt(unsigned pos);
private:
        unsigned _len;
        node<T> *_head;
};
```

ב. יוסי, הסטודנט המצטיין ב SPL הציע לחסוך עוד זיכרון ולהוריד את השדה _len ובמקומו להשתמש ב sizeof(_head). מה דעתך על הפתרון של יוסי? בחר את התשובה המתאימה ביותר (5 נקודות):

1. הפתרון מחזיר את גודל המערך, לכן עובד ונכון
2. הפתרון כמעט נכון. יש לתקן ל: sizeof(_head)/sizeof(T)
3. הפתרון מחזיר תמיד 4 במערכת של 32 ביט
4. לא יתקמפל מכיוון שלא ניתן להשתמש ב sizeof() עם template.
5. אף תשובה לא נכונה

ג. ציירו את תמונת הזכרון (stack,heap,v-table) עבור הרצת קטע הקוד הבא עד ההערה (10 נקודות):

```
template <class T>
void foo(mvector<T> vec){
        mvector<T> my_vec = vec;
        //*****here*****
}

void main(){
        mvector<int> *vec = new mvector<int>;
        int val = 0;
        vec->insertAt(val,5);
        foo(*vec);
}
```

שאלה 3 (30 נקודות)

צוות הפיתוח של חברת הזנק נחשבת החליט לעדכן את שרת ה Echo ע"פ תבנית הריאקטור שנלמדה בכיתה, כך שהתקשורת עם הלקוחות תבוסס על פרוטוקול UDP, במקום פרוטוקול TCP בקוד הקיים.

DatagramSocketChannel הינה גרסה של **DatagramSocket** המממשת גם **SelectableChannel** (כפי ש **SocketChannel** הינה גרסה של **Socket** המממשת גם **SelectableChannel**).

ניתן לייצר **DatagramSocketChannel** על **port** נתון באופן הבא:

```
int port = 7000;
DatagramSocketChannel dgChannel = DatagramSocketChannel.open();
dgChannel.socket().bind(new InetSocketAddress(port));
```

DatagramSocketChannel תומך בשליחה וקבלה של **ByteBuffer** בעזרת המתודות **send**, **receive**.
אם ה **DatagramSocketChannel** מוגדר כ **non-blocking**:
שליחה: או שכל ה **packet** נשלח מיד (כך שהערך המוחזר הוא מספר הבתים שנועדו לשליחה), או שלא נשלח כלום (כך שהערך המוחזר הוא 0).
קבלה : או שכל ה **packet** מתקבל מיד, או שלא מתקבל כלום. מתודת ה **receive** מחזירה את כתובת השולח אם כל ה **packet** נקרא אל תוך ה **ByteBuffer**, או מחזירה **null** אם לא ניתן היה לקרוא את כל ה **packet**.

להלן התיאור הרשמי של מתודות אלו:

public int send(ByteBuffer src, SocketAddress target) throws IOException
Sends a datagram via this channel.
Parameters:
src - The buffer containing the datagram to be sent
target - The address to which the datagram is to be sent
Returns: The number of bytes sent, which will be either the number of bytes that were remaining in the source buffer when this method was invoked or, if this channel is non-blocking, may be zero if there was insufficient room for the datagram in the underlying output buffer

public SocketAddress receive(ByteBuffer dst) throws IOException
Receives a datagram via this channel.
Parameters: dst - The buffer into which the datagram is to be transferred
Returns: The datagram's source address, or null if this channel is in non-blocking mode and no datagram was immediately available

א. ממשו את ההצעה של צוות הפיתוח באמצעות **DatagramSocketChannel**, כאשר **DatagramSocketChannel** יחיד משמש לתקשורת עם כל הלקוחות. קוד הריאקטור המקורי מופיע בנספח.
ניתן להוסיף מתודות ומחלקות. אין צורך לכתוב מתודות קיימות מחדש – בגיליון התשובות ציינו את המקום במתודה בו אתם עורכים שינוי ואת השינוי (20 נקודות).

ב. ענו בקצרה על השאלות הבאות:
I ציינו יתרון וחסרון עבור ההחלטה לעבור ל UDP במקום TCP (2 נקודות).

II האם ניתן לעצב את ה Reactor המקורי עם **SocketChannel** יחיד לכל הלקוחות, כפי שנעשה עם **DatagramSocketChannel** במימוש שלכם בסעיף א'? נמקו (3 נקודות).

III האם לדעתכם הגדרת **DatagramSocketChannel** נפרד לכל לקוח תייעל את התקשורת עם הלקוחות (ביחס למימוש מסעיף א' בו יש רק **DatagramSocketChannel** אחד)? נמקו (5 נקודות).

שאלה 4	(10 נקודות)

חברת הסלולר BigApple נוהגת לאחסן נתונים על לקוחותיה ועל הפעולות שהם מבצעים במכשיר הנייד שלהם.
עבור כל לקוח נשמרים נתוני שמו, מספר הטלפון שלו, תאריך הצטרפות, וכן קישורים לבני משפחתו, המנויים אף הם בחברת BigApple.
כל פעולה הקשורה ללקוח נשמרת עם תאריך הביצוע שלה. קיימים סוגים שונים של פעולות: שליחת הודעה למספר מסויים, קבלת הודעה ממספר מסויים, התקשרות למספר מסויים, שיחה שלא נענתה ממספר מסויים.

א. הגדירו מודל נתונים עבור המידע אותו שומרת חברת הסלולר.

ב. כתבו שאילתת SQL המחזירה את ההודעות שנתקבלו עבור בני משפחתו הענפה של רינגו סטאר.

נספח: קוד ה Reactor, כפי שנלמד בכיתה ובתרגול

```
public class Reactor<T> implements Runnable {

  private static final Logger logger = Logger.getLogger("edu.spl.reactor");
   private final int _port;
   private final int _poolSize;
   private final ServerProtocolFactory<T> _protocolFactory;
   private final TokenizerFactory<T> _tokenizerFactory;
   private volatile boolean _shouldRun = true;
   private ReactorData<T> _data;


 public Reactor(int port, int poolSize, ServerProtocolFactory<T> protocol, TokenizerFactory<T> tokenizer) {
      _port = port;        _poolSize = poolSize;
      _protocolFactory = protocol;        _tokenizerFactory = tokenizer;
  }

 private ServerSocketChannel createServerSocket(int port)
         throws IOException {
      try {
         ServerSocketChannel ssChannel = ServerSocketChannel.open();
         ssChannel.configureBlocking(false);
         ssChannel.socket().bind(new InetSocketAddress(port));
         return ssChannel;
      } catch (IOException e) {
         logger.info("Port " + port + " is busy");
         throw e;
      }
  }

  public void run() {
      ExecutorService executor = Executors.newFixedThreadPool(_poolSize);
      Selector selector = null;
      ServerSocketChannel ssChannel = null;

      try {
         selector = Selector.open();
         ssChannel = createServerSocket(_port);
      } catch (IOException e) {
         logger.info("cannot create the selector -- server socket is busy?");
         return;
      }
```

```
_data = new ReactorData<T>(executor, selector, _protocolFactory, _tokenizerFactory);
ConnectionAcceptor<T> connectionAcceptor = new ConnectionAcceptor<T>( ssChannel, _data);

try {
   ssChannel.register(selector, SelectionKey.OP_ACCEPT, connectionAcceptor);
} catch (ClosedChannelException e) {
   logger.info("server channel seems to be closed!");
   return;
}

while (_shouldRun && selector.isOpen()) {
   try {
      selector.select();
   } catch (IOException e) {
      logger.info("trouble with selector: " + e.getMessage());
      continue;
   }

   Iterator<SelectionKey> it = selector.selectedKeys().iterator();
   while (it.hasNext()) {
      SelectionKey selKey = (SelectionKey) it.next();
      it.remove();

      if (selKey.isValid() && selKey.isAcceptable()) {
         logger.info("Accepting a connection");
         ConnectionAcceptor<T> acceptor = (ConnectionAcceptor<T>) selKey.attachment();
         try {
            acceptor.accept();
         } catch (IOException e) {
            logger.info("problem accepting a new connection: "
                  + e.getMessage());
         }
         continue;
      }
      if (selKey.isValid() && selKey.isReadable()) {
         ConnectionHandler<T> handler = (ConnectionHandler<T>) selKey.attachment();
         logger.info("Channel is ready for reading");
         handler.read();
      }
      if (selKey.isValid() && selKey.isWritable()) {
         ConnectionHandler<T> handler = (ConnectionHandler<T>) selKey.attachment();
         logger.info("Channel is ready for writing");
         handler.write();
      }
   }
```

```
    }
    stopReactor();
}

public int getPort() {  return _port;    }

public synchronized void stopReactor() {
    if (!_shouldRun)
        return;
    _shouldRun = false;
    _data.getSelector().wakeup();
    _data.getExecutor().shutdown();
    try {
        _data.getExecutor().awaitTermination(2000, TimeUnit.MILLISECONDS);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

public static void main(String args[]) {
    if (args.length != 2) {
        System.err.println("Usage: java Reactor <port> <pool_size>");
        System.exit(1);
    }

    try {
        int port = Integer.parseInt(args[0]);
        int poolSize = Integer.parseInt(args[1]);
        Reactor<StringMessage> reactor = startEchoServer(port, poolSize);
        Thread thread = new Thread(reactor);
        thread.start();
        logger.info("Reactor is ready on port " + reactor.getPort());
        thread.join();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static Reactor<StringMessage> startEchoServer(int port, int poolSize){
    ServerProtocolFactory<StringMessage> protocolMaker = new ServerProtocolFactory<StringMessage>() {
        public AsyncServerProtocol<StringMessage> create() {
            return new EchoProtocol();
        }
    };
```

```
        final Charset charset = Charset.forName("UTF-8");
        TokenizerFactory<StringMessage> tokenizerMaker = new TokenizerFactory<StringMessage>() {
            public MessageTokenizer<StringMessage> create() {
                return new FixedSeparatorMessageTokenizer("\n", charset);
            }
        };

        Reactor<StringMessage> reactor =
                new Reactor<StringMessage>(port, poolSize, protocolMaker, tokenizerMaker);
        return reactor;
    }

        Reactor<HttpMessage> reactor = new Reactor<HttpMessage>(port, poolSize, protocolMaker,
tokenizerMaker);
        return reactor;
    }
}
```

```
public class ConnectionAcceptor<T> {
    protected ServerSocketChannel _ssChannel;

    protected ReactorData<T> _data;

    public ConnectionAcceptor(ServerSocketChannel ssChannel, ReactorData<T> data) {
        _ssChannel = ssChannel;        _data = data;
    }

    public void accept() throws IOException {
        SocketChannel sChannel = _ssChannel.accept();

        if (sChannel != null) {
            SocketAddress address = sChannel.socket().getRemoteSocketAddress();

            System.out.println("Accepting connection from " + address);
            sChannel.configureBlocking(false);
            SelectionKey key = sChannel.register(_data.getSelector(), 0);

            ConnectionHandler<T> handler = ConnectionHandler.create(sChannel, _data, key);
            handler.switchToReadOnlyMode();
        }
    }
}
```

```
public class ConnectionHandler<T> {

    private static final int BUFFER_SIZE = 1024;
```

```
    protected final SocketChannel _sChannel;
     protected final ReactorData<T> _data;
     protected final AsyncServerProtocol<T> _protocol;
    protected final MessageTokenizer<T> _tokenizer;
     protected Vector<ByteBuffer> _outData = new Vector<ByteBuffer>();
     protected final SelectionKey _skey;
     private static final Logger logger = Logger.getLogger("edu.spl.reactor");
     private ProtocolTask<T> _task = null;

    private ConnectionHandler(SocketChannel sChannel, ReactorData<T> data, SelectionKey key) {
        _sChannel = sChannel;        _data = data;        _skey = key;
        _protocol = _data.getProtocolMaker().create();
        _tokenizer = _data.getTokenizerMaker().create();
    }

    private void initialize() {
        _skey.attach(this);
        _task = new ProtocolTask<T>(_protocol, _tokenizer, this);
    }

    public static <T> ConnectionHandler<T> create(SocketChannel sChannel, ReactorData<T> data, SelectionKey
key) {
        ConnectionHandler<T> h = new ConnectionHandler<T>(sChannel, data, key);
        h.initialize();
        return h;
    }

    public synchronized void addOutData(ByteBuffer buf) {
        _outData.add(buf);
        switchToReadWriteMode();
    }

    private void closeConnection() {
        _skey.cancel();
        try {
            _sChannel.close();
        } catch (IOException ignored) {
            ignored = null;
        }
    }

    public void read() {
        if (_protocol.shouldClose())
            return;

        SocketAddress address = _sChannel.socket().getRemoteSocketAddress();
```

```
    logger.info("Reading from " + address);

    ByteBuffer buf = ByteBuffer.allocate(BUFFER_SIZE);
    int numBytesRead = 0;
    try {
        numBytesRead = _sChannel.read(buf);
    } catch (IOException e) {
        numBytesRead = -1;
    }
   if (numBytesRead == -1) {
        logger.info("client on " + address + " has disconnected");
        closeConnection();
        _protocol.connectionTerminated();
        return;
    }
     buf.flip();
    _task.addBytes(buf);
    _data.getExecutor().execute(_task);
}

public synchronized void write() {
    if (_outData.size() == 0) {
      switchToReadOnlyMode();
        return;
    }
    ByteBuffer buf = _outData.remove(0);
    if (buf.remaining() != 0) {
        try {
            _sChannel.write(buf);
        } catch (IOException e) {
            e.printStackTrace();
        }
        if (buf.remaining() != 0) {
            _outData.add(0, buf);
        }
    }
    if (_protocol.shouldClose()) {
        switchToWriteOnlyMode();
        if (buf.remaining() == 0) {
            closeConnection();
            SocketAddress address = _sChannel.socket().getRemoteSocketAddress();
            logger.info("disconnecting client on " + address);
        }
    }
}
```

```
    public void switchToReadWriteMode() {
        _skey.interestOps(SelectionKey.OP_READ | SelectionKey.OP_WRITE);
        _data.getSelector().wakeup();
    }

    public void switchToReadOnlyMode() {
        _skey.interestOps(SelectionKey.OP_READ);
        _data.getSelector().wakeup();
    }

    public void switchToWriteOnlyMode() {
        _skey.interestOps(SelectionKey.OP_WRITE);
        _data.getSelector().wakeup();
    }
}
```

```
public class ProtocolTask<T> implements Runnable {

    private final ServerProtocol<T> _protocol;
    private final MessageTokenizer<T> _tokenizer;
    private final ConnectionHandler<T> _handler;

    public ProtocolTask(final ServerProtocol<T> protocol, final MessageTokenizer<T> tokenizer, final
ConnectionHandler<T> h) {
        this._protocol = protocol;
        this._tokenizer = tokenizer;
        this._handler = h;
    }

    public synchronized void run() {
      while (_tokenizer.hasMessage()) {
        T msg = _tokenizer.nextMessage();
        T response = this._protocol.processMessage(msg);
        if (response != null) {
          try {
            ByteBuffer bytes = _tokenizer.getBytesForMessage(response);
            this._handler.addOutData(bytes);
          } catch (CharacterCodingException e) { e.printStackTrace(); }
        }
      }
    }

    public void addBytes(ByteBuffer b) {
        _tokenizer.addBytes(b);
```

```
    }
}
```

```
class FixedSeparatorMessageTokenizer implements MessageTokenizer<StringMessage> {

  private final String _messageSeparator;
   private final StringBuffer _stringBuf = new StringBuffer();
   private final Vector<ByteBuffer> _buffers = new Vector<ByteBuffer>();
   private final CharsetDecoder _decoder;
   private final CharsetEncoder _encoder;

  public FixedSeparatorMessageTokenizer(String separator, Charset charset) {
    this._messageSeparator = separator;
    this._decoder = charset.newDecoder();
    this._encoder = charset.newEncoder();
  }

  public synchronized void addBytes(ByteBuffer bytes) {
     _buffers.add(bytes);
  }

  public synchronized boolean hasMessage() {
     while(_buffers.size() > 0) {
        ByteBuffer bytes = _buffers.remove(0);
        CharBuffer chars = CharBuffer.allocate(bytes.remaining());
        this._decoder.decode(bytes, chars, false);
        chars.flip();
        this._stringBuf.append(chars);
     }
     return this._stringBuf.indexOf(this._messageSeparator) > -1;
  }

  public synchronized StringMessage nextMessage() {
    String message = null;
    int messageEnd = this._stringBuf.indexOf(this._messageSeparator);
    if (messageEnd > -1) {
      message = this._stringBuf.substring(0, messageEnd);
      this._stringBuf.delete(0, messageEnd+this._messageSeparator.length());
    }
    return new StringMessage(message);
  }

  public ByteBuffer getBytesForMessage(StringMessage msg)  throws CharacterCodingException {
    StringBuilder sb = new StringBuilder(msg.getMessage());
    sb.append(this._messageSeparator);
```

```
    ByteBuffer bb = this._encoder.encode(CharBuffer.wrap(sb));
    return bb;
  }
}
```

```
public class EchoProtocol implements AsyncServerProtocol<StringMessage> {

   private boolean _shouldClose = false;
   private boolean _connectionTerminated = false;

   public StringMessage processMessage(StringMessage msg) {
      if (this._connectionTerminated) {
         return null;
      }
      if (this.isEnd(msg)) {
         this._shouldClose = true;
         return new StringMessage("Ok, bye bye");
      }
      return new StringMessage("Your message \"" + msg + "\" has been received");
   }

  public boolean isEnd(StringMessage msg) {    return msg.equals("bye");    }

  public boolean shouldClose() {   return this._shouldClose;   }

   public void connectionTerminated() {
      this._connectionTerminated = true;
   }
}
```