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: 29.2.2016
שם המורה: ד"ר מני אדלר
ד"ר רן אטינגר
ד"ר אחיה אליסף
פרופ' אנדרי שרף
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשע"ו
סמסטר: א'
מועד: ב'
משך הבחינה: שלש שעות
חומר עזר: אסור

שאלה 1 (30 נקודות)

במועד א' התוודענו לממשק Queue המגדיר תור (FIFO) של אובייקטים.

```
interface Queue<T> {

        // Adds the given item to the queue. Throw NullPointerException if the given item is null
        void add(T item);

        // Removes the head item of the queue
        void remove();

        // Returns the head item of the queue. If the queue is empty, returns null
        T peek();

        // Returns true if this queue contains no elements
        boolean isEmpty();

        // returns the number of items in the queue
        int size();

        // returns the i-th item in the queue
        T itemAt(int i);
}
```

נתון כי המחלקה SynchQueue מממשת את הממשק Queue עם סנכרון מלא (=כל המתודות מסונכרנות על this, כל השדות private).

המחלקה ExIntQueue מרחיבה את המחלקה SynchQueue על ידי הוספה של מתודה sum() המחזירה את סכום המספרים בתור.

```
class ExtIntQueue extends SynchQueue<Integer> {
        public int sum() {
                int s=0;
                for (int i=0; i<size(); i++)
                        s += peek(i);
                return s;
        }
}
```

א. הגדירו תנאי התחלה וסיום למתודה [4 נקודות]

ב. תארו תרחיש בו לא מתקיימים תנאי הסיום [6 נקודות]

ג. עדכנו את המחלקה ע"פ גישת optimistic try and fail שנלמדה בכיתה (כמו לדוגמא ה versioned iterator), כך שתנאי הסיום יישמרו. בכל מקרה, אין לסנכרן את המתודה לכל משך ביצוע הסכימה. [10 נקודות].

בעיית 'המיסטיקנים הרעבים' דומה לבעיית הפילוסופים הרעבים, בשינוי קל: כדי לכוון את מחשבותיו, המיסטיקן מתבונן בחפץ כלשהו (לדוגמא לשימוש בפרקטיקה זו עם שעון, ראו: 'דרך המלך' עמ' ת"נ- תנ"א).
בשאלה זו נעסוק במיסטיקנים המתבוננים במזלגות המונחים על השולחן מימינם ומשמאלם בזמן שהם חושבים.

להלן הקוד של בעיית 'המיסטיקנים הרעבים' (השינוי ממועד א', במתודת ה think, מסומן במרקר).

```
class Semaphore {
   private final int _permits;
   private int _free;

   public Semaphore(int permits) {
      _permits = permits;
      _free = permits;
   }

   public synchronized void acquire() throws InterruptedException {
      while (_free<=0)
         wait();
      _free --;
   }

   public synchronized void release() throws InterruptedException {
      if (_free < _permits) {
```

```
        _free++;
        notifyAll();
      }
    }
}
```

```
class Mystic implements Runnable  {

    protected final int _id;
    protected final Semaphore _leftFork;
    protected final Semaphore _rightFork;

    public Philosopher(int id, Semaphore leftFork, Semaphore rightFork) {
            _id = id;
            _leftFork = leftFork;
            _rightFork = rightFork;
    }

    public void run() {
      try {
            while (true){
                        think();
                        eat();
            }
      } catch (InterruptedException e) {  return;  }
    }

    protected void think() {

            Semaphore firstFork = (_id % 2 == 0 ? _leftFork : _rightFork);
            Semaphore secondFork = (_id % 2 == 0 ? _rightFork : _leftFork);

             // 1. acquire the forks
             firstFork.acquire();
             secondFork.acquire();

             // 2. Think (the 'thinking' is simulated by a busy wait)
            for (int i=0; i< 100000; i++);

            // 3. release the forks
            secondFork.release();
            firstFork.release();
```

```
  }


  protected void eat() throws InterruptedException {
          Semaphore firstFork = (_id % 2 == 0 ? _leftFork : _rightFork);
          Semaphore secondFork = (_id % 2 == 0 ? _rightFork : _leftFork);

          // 1. acquire the forks
          firstFork.acquire();
          secondFork.acquire();

         // 2. eat (the 'eating' is simulated by a busy wait)
         for (int i=0; i< 100000; i++);

         // 3. release the forks
         secondFork.release();
         firstFork.release();
  }
}
```

```
class HungrayMystics {

  public static void main(String [] args){

     Semaphore fork1 = new Semaphore(1);
     Semaphore fork2 = new Semaphore(1);
     Semaphore fork3 = new Semaphore(1);

     Mystic mystic1 = new Mystic (1,fork1,fork2);
     Mystic mystic 2 = new Mystic (2,fork2,fork3);
     Mystic mystic 3 = new Mystic (3,fork3,fork1);

     new Thread(mystic1).start();
     new Thread(mystic2).start();
     new Thread(mystic3).start();
  }
}
```

ד. עדכנו את מחלקת ה Semaphore (ניתן לשנות את הקריאות למתודות המחלקה ממחלקות אחרות), כך שמיסטיקנים שחושבים (מתודת think) יוכלו להתבונן ביחד באותה מזלג, בתנאי שאף אחד לא אוכל בה. אך מצד שני, לא ניתן יהיה לחשוב על מזלג בזמן שמיסטיקן אחר אוכל בה (וכל שכן שלא לאכול עם מזלג שמיסטיקן אחר אוכל בה) [10 נקודות]

שאלה 2 (30 נקודות)

בשאלה זו נעסוק בפיתוח מבנה כללי לאיחסון אוביקטים שנקרא ITEM_COLLECTION.

```
class SHAPE{
public:
        virtual SHAPE* Clone() = 0;
};
class SPHERE: public SHAPE{
        virtual SHAPE* Clone() { …}
};
class BOX : public SHAPE{
        virtual SHAPE* Clone() { …}
};
template <class T> class ITEM_COLLECTION{
public:
        ITEM_COLLECTION();
        ITEM_COLLECTION(int n)  { /*הניחו כי קיים מימוש תקין המקצה תאים*/ }
        ITEM_COLLECTION(const ITEM_COLLECTION &s);
        ~ITEM_COLLECTION() { Clear();}
        void Clear(){
           for (int i = 0; i < _size; i++)
              delete _items[i];
           delete _items;
           _items = 0;
        }
        void Add(T item, int i) { _items[i] = item; }

        void byVal(ITEM_COLLECTION<T> s) {
             //do something
        }

private:
        int _size;
        T *_items;
};
```

א. ממשו בנאי מעתיק ITEM_COLLECTION(const ITEM_COLLECTION &s) כך שתבוצע העתקה עמוקה. יש לשים לב לשימוש נכון בסינטקס של template.[10 נקודות]

ב. סטודנט כתב את הקוד הבא המשתמש במבנה האיחסון הנ"ל:

```
void main(){
```

```
1.  ITEM_COLLECTION</*here*/> myShapeCollection(50);
2.  for (int i = 0; i < 50; i++)    {
3.          /*here*/
4.          myShapeCollection.Add(/*here*/,i); //adds a SPHERE
5.  }
6.  byVal(myShapeCollection);
}
```

השלימו את שורות 1-4 (/*here*/). יש לדאוג לכך שניתן יהיה לאחסן במבנה גם אוביקטים מסוג SPHERE וגם אובייקטים מסוג BOX [5 נקודות]

ג. בעקבות ביטול הדרישה להעתקה עמוקה, בחר הסטודנט להשתמש בבנאי המעתיק, הניתן כברירת מחדל (default constructor), של **ITEM_COLLECTION** (במקום זה שמומש בסעיף א').
איזו בעיה נוצרת בעקבות הקריאה לפונקציה **byVal(myShapeCollection);** בשורה 6? [5 נקודות]

ד. כדי לפתור את הבעיה מסעיף ג', החליט הסטודנט להשתמש במחלקה **shared_ptr** (או **smart_ptr**) המנהלת מצביע כפי שנלמד בכיתה. הציעו לסטודנט פתרון בעזרת **shared_ptr** (שורות 1-4). אין לשנות את המחלקה **ITEM_COLLECTION**. [10 נקודות]

שאלה 3 (30 נקודות)

אחד הסטודנטים בקורס ציין כי הקוד הנוכחי של הריאקטור אינו נותן עדיפות ללקוחות שהטרידו פחות את השרת בתקשורת. יתירה מזאת, בקוד הנוכחי אין בהכרח עדיפות ללקוח שמספר הבקשות שלו שטופלו עד כה נמוך יחסית (בקשה = הודעה שלמה, ע"פ הפרוטוקול, לביצוע).

א. הציגו תרחישים המדגימים את טענתו הנכונה של הסטודנט (<u>כל תרחיש במשפט קצר אחד</u>):
- תרחיש בו הלקוח הנבחר לטיפול ב **executor** הינו הלקוח שמספר ה bytes שנתקבלו ממנו ונשלחו אליו הוא הגבוה ביותר.
- תרחיש בו הלקוח הנבחר לטיפול ב **executor** הינו הלקוח שמספר הבקשות הקודמות שלו שטופלו הינו הגבוה ביותר. [10 נקודות]

ב. ממשו בשרת את שתי המדיניויות למתן עדיפות (<u>אין להעתיק מחדש קוד. רק לציין היכן ומה השינוי</u>):
- מדיניות בה המשימה הנבחרת לביצוע ב **executor** היא זו של הלקוח שמספר הבתים הכולל שהתקבלו ממנו ונשלחו אליו עד כה הוא הנמוך ביותר.
- מדיניות בה המשימה הנבחרת לביצוע ב **executor** היא זו של הלקוח שמספר הבקשות הקודמות שלו שטופלו, ביחס למספר הכולל של הבקשות שטופלו עד כה בשרת, הוא הנמוך ביותר. [20 נקודות]

<u>חומר עזר</u>:

- הקוד של הריאקטור, כפי שנלמד בכיתה, ניתן בנספח של המבחן.

- המחלקה **PriorityBlockingQueue** מרחיבה את המחלקה **BlockingQueue**, כך שמתודת ה **remove()** לא מחזירה את הת'רד שהוכנס ראשון, אלא את הת'רד בעל סדר העדיפות הגבוה ביותר ע"פ קרטריון מתודת ה **compareTo()** של האובייטים בתור, המבוצעת במסגרת הפעלת מתודת ה **enqueue** של התור.

- מימוש מחלקת ה Executor:

```
public class Executor {

   private BlockingQueue<Runnable> taskQueue;

   public Executor(int numOfThreads) {
      taskQueue = new BlockingQueue<Runnable>();
      for(int i=0; i< numOfThreads; i++)
            new Thread(new PoolThread(taskQueue)).start();
   }

   public synchronized void  execute(Runnable task) throws Exception{
      taskQueue.enqueue(task);
   }
}
```

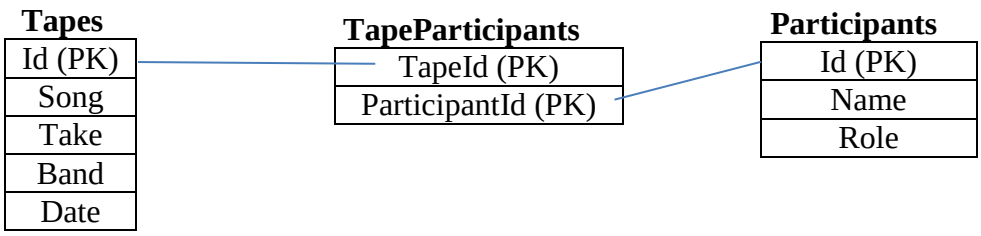
```
public class PoolThread extends Runnable {

   private BlockingQueue<Runnable> taskQueue;

   public PoolThread (BlockingQueue<Runnable> taskQueue)   {
            this.taskQueue = taskQueue;
   }

   public void run(){
      while(true) {
         try{
            taskQueue.dequeue().run();
         } catch(Exception e)  { }
      }
   }
}
```

שאלה 4 (10 נקודות)

במועד א' בנינו מודל נתונים רלציוני עבור נתוני ההקלטות באולפני Abbey Road בלונדון:

סרט הקלטה מאופיין ע"י תאריך, שיר, שם הלהקה, רשימת המשתתפים בהקלטה, ומספר ה'טייק' (כל שיר מוקלט בדרך כלל מספר רב של פעמים עד לתוצאה הרצויה, כל ניסיון שכזה מכונה take. כל סרט הקלטה מכיל טייק אחד). בהקלטה עשויים להשתתף אנשים בעלי תפקידים שונים: נגנים (נגן מסויים יכול לנגן באותה הקלטה במספר כלים שונים), זמרים וטכנאי הקלטה.

להלן המודל שהוצע בפתרון המבחן :



א. הדגימו כיצד ניתן לכלול במשתתפים בהקלטות באולפן את קים פילבי בתפקיד סוכן הק.ג.ב. הכפול. [2 נקודות]

ב. הרחיבו את מודל הנתונים כך שניתן יהיה להזין לנתוני ההקלטה אך ורק משתתפים בתפקידים הבאים: זמר, גיטריסט, בסיס, קלידן, מתופף, טכנאי הקלטות, מפיק מוזיקלי. [4 נקודות]

ג. כתבו שאילתת SQL המחזירה את שמות שירי הביטלס אותם הפיק 'פיל ספקטור', ממויינים ע"פ מספר הטייק. (לא ניתן להסתמך על העובדה שפיל ספקטור לקח ב 1970 את ההקלטות של get back project והפיק מהן את האלבום let it be)
[4 נקודות]

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;
   }
}
```