אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

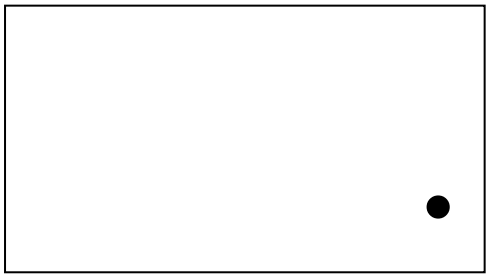
רשמו תשובותיכם בגיליון התשובות בלבד
תשובות מחוץ לגיליון לא יבדקו.

בהצלחה!

תאריך הבחינה: 11.2.2013
שם המורה: ד"ר אנדרי שרף
ד"ר מני אדלר
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשע"ג
סמסטר: א'
מועד: א'
משך הבחינה: שלש שעות
חומר עזר: אסור

שאלה 1 (30 נקודות)

הממשק BlockedBall מייצג כדור במרחב דו ממדי, הנתון בתוך מסגרת מלבנית, כך שהכדור אינו יכול לחרוג ממסגרת זו.



```
interface BlockedBall {
        int getRectUpper(); //returns the upper line of the blocking rectangle
        int getRectBottom(); //returns the bottom do line of the blocking rectangle
        int getRectLeft(); //return the left line of the blocking rectangle
        int getRectRight(); //return the right line of the blocking rectangle

        int getBallX(); //returns the X position of the ball
        int getBallY(); //returns the Y position of the ball

        void up() throws Exception; // increases the Y position by 1
        void down() throws Exception; //decreases the Y position by 1
        void right() throws Exception; // increases the X position by 1
        void left() throws Exception; //decreases the X position by 1
}
```

א. הגדירו את התכונה נשמרת (invariant) עבור הממשק BlockedBall, ואת תנאי ההתחלה והסיום למתודות up() down() left() right(), וממשו את המתודות במחלקה CheckedBlockedBall בהתאם [12 נקודות]

```
abstract class CheckedBlockedBall implements BlockedBall {

	public boolean checkInv() {
		// returns true if the invariant holds
		// @TODO
	}

	public boolean checkPreCondUp() {
		// returns true if the pre-condition of up() method holds
		// @TODO
	}

	public boolean checkPreCondDown() {
		// returns true if the pre-condition of down() method holds
		// @TODO
	}

	public boolean checkPreCondRight() {
		// returns true if the pre-condition of right() method holds
		// @TODO
	}

	public boolean checkPreCondLeft() {
		// returns true if the pre-condition of left() method holds
		// @TODO
	}

}
```

להלן מימוש של הממשק **BlockedBall**:

```
class SimpleBlockedBall extends CheckedBlockedBall {

	SimpleBlockedBall(int rectUp, int rectBottom, int rectLeft, int rectRight, int ballX, int ballY) throws
Exception {
		_rectUp = rectUp; _rectBottom = rectBottom; _rectLeft = rectLeft; _rectRight = rectRight;
		_ballX = ballX; _ballY = ballY;
		if (!checkInv())
			throw new Exception("Invariant does not hold!");
```

```
        }

        public int getRectUpper() { return _rectUp; }
        public int getRectBottom() { return _rectBottom; }
        public int getRectLeft() { return _rectLeft; }
        public int getRectRight() { return _rectRight; }
        public int getBallX() { return _ballX; }
        public int getBallY() { return _ballY;  }

        public void up() throws Exception {
                if (!checkPreCondUp())
                        throw new Exception("The pre-condition for the up() method does not hold!");
                _ballY++;
        }
        public void down() throws Exception {
                if (!checkPreCondDown())
                        throw new Exception("The pre-condition for the down() method does not hold!");
                _ballY--;
        }

        public void right() throws Exception {
                if (!checkPreCondRight())
                        throw new Exception("The pre-condition for the right() method does not hold!");
                _ballX++;
        }
        public void left() throws Exception {
                if (!checkPreCondLeft())
                        throw new Exception("The pre-condition for the left() method does not hold!");
                _ballX--;
        }

        protected final int _rectUp, _rectBottom, _rectLeft, _rectRight;
        protected int _ballX, _ballY;
}
```

בתוכנית הבאה נע הכדור בתוך המלבן החוסם אותו. התנועה של הכדור נקבעת על פי שני כוחות המופעלים עליו, האחד בכיוון מעלה/מטה והשני בכיוון ימין/שמאל.
בעקבות פגיעה בכל אחת ממסגרות המלבן משתנה כיוון התנועה של הכדור:

- פגיעה בקו התחתון: הכדור משנה את כיוונו ועולה למעלה
- פגיעה בקו עליון: הכדור משנה את כיוונו ויורד למטה
- פגיעה בקו השמאלי: הכדור משנה את כיוונו לצד ימין
- פגיעה בקו הימני: הכדור משנה את כיוונו לצד שמאל

שני כוחות אלו מסומלצים על ידי הת'רדים HorizontalMovment, VerticalMovment:

```
enum Orientation  {
	LEFT, RIGHT, UP, DOWN ;
}
```

```
class VerticalMovment implements Runnable {

	BlockedBall _blockedBall;
	Orientation _orientation;

	VerticalMovment(BlockedBall blockedBall, Orientation orientation) {
		_blockedBall = blockedBall;
		_orientation = orientation;
	}

	public void run() {
		while (!Thread.interrupted()) {
			int y = _blockedBall.getBallY();
			if (y == _blockedBall.getRectUpper())
				_orientation = Orientation.DOWN;
			if (y == _blockedBall.getRectBottom())
				_orientation = Orientation.UP;

			try {
				if (_orientation == Orientation.UP)
					_blockedBall.up();
				else
					_blockedBall.down();
			} catch (Exception e) {
				System.out.println(e);
			}
		}
	}
}
```

```
class HorizontalMovment implements Runnable {

	BlockedBall _blockedBall;
	Orientation _orientation;

	HorizontalMovment(BlockedBall blockedBall, Orientation orientation) {
		_blockedBall = blockedBall;
```

```
                _orientation = orientation;
        }

        public void run() {
                while (!Thread.interrupted()) {
                        int x = _blockedBall.getBallX();
                        if (x == _blockedBall.getRectLeft())
                                _orientation = Orientation.RIGHT;
                        if (x == _blockedBall.getRectRight())
                                _orientation = Orientation.LEFT;

                        try {
                                if (_orientation == Orientation.LEFT)
                                        _blockedBall.left();
                                else
                                        _blockedBall.right();
                        } catch (Exception e) {
                                System.out.println(e);
                        }
                }
        }
}
```

```
class Simulation  {
        public static void main(String[] args) throws Excepption {
                BlockedBall blockedBall = new SimpleBlockedBall(10,1,1,10,5,5);
                new Thread(new HorizontalMovment(blockedBall, Orientation.LEFT)).start();
                new Thread(new VerticalMovment(blockedBall, Orientation.UP)).start();
        }
}
```

ב. האם הרצת התוכנית בטוחה? נמקו בקצרה [5 נקודות]

לסימולציה נוסף כעת כוח נוסף – הרוח. הרוח עשויה בכל שלב להזיז את הכדור למעלה או למטה, ימינה או שמאלה. בקוד שלהלן מסומלצת הרוח ע"י ת'רד שלישי במערכת, Wind שמו.

```
public class Wind implements Runnable {

        BlockedBall _blockedBall;
        List<Orientation> _orientations;
        Random _rand;

        Wind(BlockedBall blockedBall) {
```

```
		_blockedBall = blockedBall;
		_orientations = new ArrayList<Orientation>();
		_rand = new Random();
	}

	public void run() {
		while (!Thread.interrupted()) {
			Orientation orientation = _orientations.get(_rand.nextInt(4));
			try {
				if (orientation == Orientation.LEFT)
					_blockedBall.left();
				if (orientation == Orientation.RIGHT)
					_blockedBall.right();
				if (orientation == Orientation.UP)
					_blockedBall.up();
				if (orientation == Orientation.DOWN)
					_blockedBall.down();
			} catch (Exception e) {
			}
		}
	}
}
```

```
public class Simulation {
	public static void main(String[] args) throws Exception {
		BlockedBall blockedBall = new SimpleBlockedBall(10,1,1,10,5,5);
		new Thread(new HorizontalMovment(blockedBall, Orientation.LEFT)).start();
		new Thread(new VerticalMovment(blockedBall, Orientation.UP)).start();
		new Thread(new Wind(blockedBall)).start();
	}
}
```

ג. הסבירו בקצרה מדוע המערכת אינה בטוחה כעת [5 נקודות]

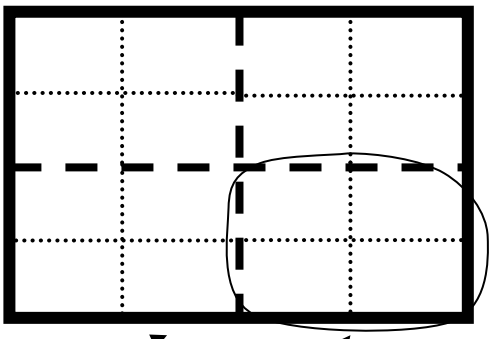
ד. עדכנו את המחלקה SimpleBlockedBall (בלבד), כך שהרצת הסימולציה עם שלושת הת'רדים תהיה בטוחה. במימוש שלכם חייב לאפשר הזזה במקביל של הכדור על ציר X (ימינה או שמאלה) ועל ציר Y (למעלה או למטה). [8 נקודות]

שאלה 2 (30 נקודות)

נדרש כעת לממש מבנה חיפוש מהיר ב C++ עבור הBlockedBall משאלה 1. המבנה שהוחלט עליו הינו חלוקה היראכרכית של הRectangle לתת-מלבנים באופן רקורסיבי.
להלן מימוש חלקי של המבנה:

```
class SceneCell{
public:
        virtual  SceneCell* retrieveCell(float x, float y)=0;
        virtual  void subdivide(int height) = 0;
};

class RectangleCell : public SceneCell{
public:
        RectangleCell(float left, float bottom, float right, float top);
        virtual ~RectangleCell();
        virtual  SceneCell* retrieveCell(float x, float y);
        virtual  void subdivide(int height);
private:
        float _left, _right, _top, _bottom; //rectangle limits
        RectangleCell *_topLeftSon, *_topRightSon, *_bottomLeftSon, *_bottomRightSon; //rectangle cell sons
};
```



א. ממשו את הפונקציות subdivide ו retrieveCell.

subdivide מקבלת את גובה המבנה ומפצלת את ה RectangleCell ל 4 בנים (_topLeftSon, _topRightSon, _bottomLeftSon, _bottomRightSon) באופן רקורסיבי לפי העומק הנדרש (עומק 0 הוא עבור מלבן יחיד), כאשר הבנים הם חלוקה של מלבן האב ל4 תת-מלבנים, כמו בציור.
שימו לב – אין להשתמש בלולאה אלא ברקורסיה.
retrieveCell מחזירה את התא הקטן ביותר המכיל את הנקודה (x,y), או null אם הנקודה מחוץ למלבן.
יש לממש ביעילות.
[12 נקודות]

ב. מה הבעיה בקטע קוד הבא? עדכנו את המימוש בהתאם [6 נקודות]

```
void main(){
    RectangleCell rcell(0.0, 0.0, 2.0, 2.0);
    rcell.subdivide(17);
}
```

ג. כתבו את תמונת הזיכרון המתקבלת בסימון here [12 נקודות]

```
void foo(SceneCell& s1, RectangleCell r1){

      SceneCell *s2 = r1.retrieveCell(1,1);

      s1.subdivide(1);

      s2->subdivide(1);

     //----here-------
}

void main(){

      int height = 1;

      RectangleCell rcell(0.0,0.0,2.0,2.0);

      rcell.subdivide(height);

      SceneCell *scell = new RectangleCell(0.0,0.0,2.0,2.0);

      foo(*scell, rcell);

}

void foo(SceneCell& s1, RectangleCell r1){

      SceneCell *s2 = r1.retrieveCell(1,1);

      s1.subdivide(1);

      s2->subdivide(1);

     //----here-------
}

void main(){

      int height = 1;

      RectangleCell rcell(0.0,0.0,2.0,2.0);

      rcell.Subdivide(height);

      SceneCell *scell = new RectangleCell(0.0,0.0,2.0,2.0);

      foo(*scell, rcell);

}
```

שאלה 3 (20 נקודות)

בנספח למבחן מופיע קוד ה Reactor כפי שנלמד בכיתה ובתרגול.

על מחשב עם 24 מעבדים בוצעה שורת הפקודה הבאה:

```
>java Reactor 7895 12
```

נתון כי שני לקוחות מחוברים לשרת, וכי לקוחות אלו שלחו את ההודעות הבאות:

לקוח א: That was can-you-dig-it by Georgie Wood \n

לקוח ב: And now we'd like to do ark-the-angels-come \n

נתון כי הודעות אלו נקראו במלואן מה socket של כל לקוח ע"י ה ConnectionHandler שלו, והועברו ל Executor

א. תארו תרחיש בו, על אף שיש 24 מעבדים, רק ת'רד אחד מתוך 12 הת'רדים ב Executor נמצא במצב ready, ואילו כל השאר נמצאים במצב blocked, למרות שתור המשימות ב Executor אינו ריק. [5 נקודות]

ב. סטודנט אחד הציע לפתור את התרחיש הבעייתי על ידי שינוי הסנכרון במתודה run() של המחלקה ProtocolTask:

```
public void run() {
    while (_tokenizer.hasMessage()) {
      synchronized(this) {
        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(); }
        }
      }
    }
}
```

האם לדעתכם הצעת הסטודנט פותרת את הבעיה? [5 נקודות]

ג. סטודנטית אחרת הציעה לפתור את התרחיש הבעייתי מהסעיף הקודם, על ידי צמצום מספר ה ProtocolTask של כל לקוח בתור המשימות של ה Executor לאחד. ממשו את הצעתה [10 נקודות]

חומר עזר: המתודות של המחלקה ThreadPoolExecutor, המימוש הסטנדרטי של ExecutorService (= ניתן להניח שהמתודה Executors.newFixedThreadPool מחזירה מופע של מחלקה זו, או של מחלקה היורשת אותה)

Method Summary	
protected void	**afterExecute**(Runnable r, Throwable t) Method invoked upon completion of execution of the given Runnable.
void	**allowCoreThreadTimeOut**(boolean value) Sets the policy governing whether core threads may time out and terminate if no tasks arrive within the keep-alive time, being replaced if needed when new tasks arrive.
boolean	**allowsCoreThreadTimeOut**() Returns true if this pool allows core threads to time out and terminate if no tasks arrive within the keepAlive time, being replaced if needed when new tasks arrive.
boolean	**awaitTermination**(long timeout, TimeUnit unit) Blocks until all tasks have completed execution after a shutdown request, or the timeout occurs, or the current thread is interrupted, whichever happens first.
protected void	**beforeExecute**(Thread t, Runnable r) Method invoked prior to executing the given Runnable in the given thread.
void	**execute**(Runnable command) Executes the given task sometime in the future.
protected void	**finalize**() Invokes shutdown when this executor is no longer referenced.
int	**getActiveCount**() Returns the approximate number of threads that are actively executing tasks.
long	**getCompletedTaskCount**() Returns the approximate total number of tasks that have completed execution.
int	**getCorePoolSize**() Returns the core number of threads.
long	**getKeepAliveTime**(TimeUnit unit) Returns the thread keep-alive time, which is the amount of time that threads in excess of the core pool size may remain idle before being terminated.
int	**getLargestPoolSize**() Returns the largest number of threads that have ever simultaneously been in the pool.
int	**getMaximumPoolSize**() Returns the maximum allowed number of threads.
int	**getPoolSize**() Returns the current number of threads in the pool.
BlockingQueue<Runnable>	**getQueue**() Returns the task queue used by this executor.
RejectedExecutionHandler	**getRejectedExecutionHandler**() Returns the current handler for unexecutable tasks.
long	**getTaskCount**() Returns the approximate total number of tasks that have ever been scheduled for

	execution.
ThreadFactory	**getThreadFactory**() Returns the thread factory used to create new threads.
boolean	**isShutdown**() Returns true if this executor has been shut down.
boolean	**isTerminated**() Returns true if all tasks have completed following shut down.
boolean	**isTerminating**() Returns true if this executor is in the process of terminating after shutdown or shutdownNow but has not completely terminated.
int	**prestartAllCoreThreads**() Starts all core threads, causing them to idly wait for work.
boolean	**prestartCoreThread**() Starts a core thread, causing it to idly wait for work.
void	**purge**() Tries to remove from the work queue all Future tasks that have been cancelled.
boolean	**remove**(Runnable task) Removes this task from the executor's internal queue if it is present, thus causing it not to be run if it has not already started.
void	**setCorePoolSize**(int corePoolSize) Sets the core number of threads.
void	**setKeepAliveTime**(long time, TimeUnit unit) Sets the time limit for which threads may remain idle before being terminated.
void	**setMaximumPoolSize**(int maximumPoolSize) Sets the maximum allowed number of threads.
void	**setRejectedExecutionHandler**(RejectedExecutionHandler handler) Sets a new handler for unexecutable tasks.
void	**setThreadFactory**(ThreadFactory threadFactory) Sets the thread factory used to create new threads.
void	**shutdown**() Initiates an orderly shutdown in which previously submitted tasks are executed, but no new tasks will be accepted.
List<Runnable>	**shutdownNow**() Attempts to stop all actively executing tasks, halts the processing of waiting tasks, and returns a list of the tasks that were awaiting execution.
protected void	**terminated**() Method invoked when the Executor has terminated.

שאלה 4 (10 נקודות)

ציינו נכון/לא נכון על הקביעות הבאות:

א. תהליך המעונין להתחבר ל RemoteObject חייב לדעת את ה host של התהליך בו מוגדר בזיכרון ה RemoteObject.
ב. תקשורת בין תהליכים על בסיס מתודות של RemoteObject אמינה יותר (מבחינת הבטחת הגעת המידע כסִדְרו) מהעברת הודעות ב sockets של TCP.
ג. הערך החוזר ממתודה של RemoteObject חייב לממש את הממשק Serializable.
ד. אלגוריתם Selective Repeat שומר על סדר קבלת ההודעות.
ה. באלגוריתם Stop & Wait אין צורך למספר את ההודעות.

שאלה 5 (10 נקודות)

גניזת קהיר היא אוסף גדול של כתבי יד (כלומר, דף או מגילה עתיקים עם תוכן טקסטואלי), שנכתבו בין המאה ה-9 והמאה ה-19, ונשמרו בגניזה בעליית הגג של בית הכנסת בן עזרא בקהיר. כתבים אלו – פיוטים, מסמכים משפטיים, רשימות שמיות, קטעי פנקסים, מכתבים, ועוד - מלמדים רבות על התרבות היהודית במצרים, ארץ ישראל ואגן הים התיכון.

פרויקט 'גנזים' הינו מפעל רחב היקף אשר מטרתו לסרוק באיכות גבוהה את כל קטעי הגניזה (כחצי מיליון מסמכים) ולהנגישו ברשת לציבור החוקרים ולשוחרי הדעת.

במסגרת הפרוייקט הוקם מסד נתונים המכיל פרטים שונים על כל אחד מכתבי היד:
- מספר קטלוגי של כתב היד
- הנושא בו עוסק כתב היד
- קישור לסריקה של כתב היד
- פרטי המחברים של כתב היד, אם הם ידועים: שם, תאריך לידה, מקום מגורים

א. הגדירו מודל נתונים (טבלאות ומפתחות) עבור המידע הנ"ל [4 נקודות]

ב. כתבו שאילתת SQL המחזירה את הקישורים לכתבי היד הסרוקים העוסקים בפיוט, ממוינים בסדר עולה ע"פ המספר הקטלוגי, ובמידה וידוע מי המחברים שלהם, גם את פרטי המחברים [6 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;
    }

    public static Reactor<HttpMessage> startHttpServer(int port, int poolSize) throws Exception{
        ServerProtocolFactory<HttpMessage> protocolMaker = new ServerProtocolFactory<HttpMessage>() {
            public AsyncServerProtocol<HttpMessage> create() {
                return new HttpProtocol();
            }
        };

        TokenizerFactory<HttpMessage> tokenizerMaker = new TokenizerFactory<HttpMessage>() {
            public MessageTokenizer<HttpMessage> create() {
                return new HttpMessageTokenizer();
            }
        };

        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;
   }
}
```