אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

רשמו תשובותיכם בגיליון התשובות בלבד
תשובות מחוץ לגיליון לא יבדקו.

בהצלחה!

תאריך הבחינה: 13.2.2012
שם המורה: פרופ' מיכאל אלחדד
ד"ר מני אדלר
ד"ר אנדרי שרף
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשע"ב
סמסטר: א'
מועד: א'
משך הבחינה: שלש שעות
חומר עזר: אסור

שאלה 1 (30 נקודות)

הממשק Circle מייצג מעגל במרחב דו ממדי, תחת האילוץ שהקורדינטות של כל הנקודות במעגל הינן חיוביות (במילים אחרות, המעגל ממוקם ברבע החיובי של הצירים Y,X, בו ערכי x ו y גדולים או שווים ל 0). המעגל מיוצג על ידי קורדינטות נקודת המרכז, ואורך הרדיוס:

```
interface Circle {
  int getX();     // returns the x coordinate of the circle center point
  int getY();     // returns the y coordinate of the circle center point
  int getRadius();  // returns the length of the circle radius
  void setCenter(int x, int y)   throws Exception;  // set the coordinates for the circle center point
  void setRadius(int radius)  throws Exception;  // set the length of the circle radius
}
```

א. הגדירו תכונה נשמרת (invariant) עבור הממשק Circle [5 נקודות]

להלן מימוש של הממשק Circle:

```
class CircleImpl implements Circle {
  int _x;
  int _y;
  int _radius;

  CircleImpl(int x, int y, int radius) {
    _x = x;
    _y = y;
    _radius = radius;
  }

  public int getX() {
```

```
    return _x;
  }
  public int getY() {
    return _y;
  }
  public int getRadius() {
    return _radius;
  }

  public void setCenter(int x, int y)    throws Exception {
    _x = x;
    _y = y;
  }
  public void setRadius(int radius)  throws Exception  {
    _radius = radius;;
  }
}
```

ב. עדכנו את המחלקה **CircleImpl** כך שתהא בטוחה בהרצה סדרתית (אין לדאוג בסעיף זה לבטיחות בהרצה מקבילית) [3 נקודות]

ג. הסבירו מדוע המחלקה עדיין אינה בטוחה תחת הרצה מקבילית, ועדכנו אותה בהתאם כך שתשמור על בטיחות בכל סוג הרצה שכזה [4 נקדות]

ד. נניח כי הוסרה המגבלה על מיקום המעגל ברבע החיובי של הצירים, כך שהמעגל יכול להיות ממוקם בכל מקום.

i. נסחו מחדש את התכונה הנשמרת לממשק [2 נקודות]
ii. נמקו האם המימוש להרצה סדרתית בסעיף ב' בטוח כעת גם עבור הרצה מקבילית. [3 נקודות]
iii. נמקו האם המימוש להרצה סדרתית בסעיף ב' נכון עבור הרצה מקבילית (מבחינת התאמת הפלט המקבילי לפלט סדרתי בסדר כל שהוא). [5 נקודות]
iv. עדכנו את המחלקה כך שתשמור על הבטיחות והנכונות תחת כל הרצה מקבילית. במימוש הפתרון עליכם לאפשר ביצוע במקביל של עדכון נקודת המרכז ושל עדכון אורך הרדיוס. [8 נקודות]

שאלה 2 (30 נקודות)

א. כתבו את הממשק Circle ב C++ (תנו את הדעת לאופן ייצוג ממשק ב C++) [3 נקודות]

ב. הוסיפו לממשק את המתודות הבאות: [2 נקודות]
getNeighbours() המחזירה רשימה של מצביעים למעגלים הסמוכים למעגל הנתון
addNeighbour(Circle*) הוספת מעגל שכן
removeNeighbour(Circle*) הסרת מעגל שכן

ג. כתבו את המחלקה CircleImpl המממשת את הממשק Circle כנדרש ב C++. [12 נקודות]

ד. נתון קטע הקוד הבא ב++C

```
CircleImpl dummy(const CircleImpl & c1) {
   CircleImpl c2;
   c2 = c1
  //@@ יש לצייר את תמונת הזיכרון כאשר הרצת הקוד מגיעה לכאן
   return c2;
}
void main() {
  CircleImpl c1(5, 3, 2);
  CircleImpl c2(c1);
  CircleImpl *c3 = new CircleImpl();

  c3 = dummy(c1);

}
```

ציירו את תמונת הזיכרון, כאשר מריצים את **main** ומגיעים למקום המסומן בתוך קוד המתודה **dummy** [13 נקודות].

שאלה 3 (30 נקודות)

בנספח למבחן מופיע החלק המרכזי של קוד ה **Reactor** כפי שנלמד בכיתה ובתרגול.

א. עדכנו את הקוד, כך שהמחלקה **Reactor** תממש את הממשק **Counting** [10 נקודות]

```
interface Counting {
  int getNumBytesRead();  // returns the total  number of bytes that were read from all clients
  int getNumBytesWrite();  // returns the total  number of bytes that were written to all clients
  int getFailedAccepts ();  // returns the total  number client accepts that were failed
}
```

המתודה **getBytesRead()** מחזירה את מספר הבתים אשר נקראו עד כה מכל הלקוחות
המתודה **getBytesWrite()** מחזירה את מספר הבתים אשר נכתבו עד כה לכל הלקוחות
המתודה **getFailedAccepts()** מחזירה את מספר הניסיונות שנכשלו בקבלת לקוח חדש

ב. נתונה המחלקה **SearchService** המכילה את המתודה **search(word)**. מתודה זו מקבלת מילה לחיפוש ומחזירה רשימה של שמות קבצים בהם המילה מופיעה.
המחלקה נעזרת באוסף של קבצי אינדקס המכילים עבור כל מילה את רשימת הקבצים הנדרשת. כל קובץ אינדקס מכיל את רשימות הקבצים עבור מילים המתחילות באות מסוימת. כך שהמתודה search צריכה לבחור את קובץ האינדקס המתאים, על פי האות הראשונה במילה המבוקשת, ואחר כך לסרוק את הקובץ עד המילה המבוקשת, כדי לקבל את רשימת הקבצים בהם היא מופיעה.
להלן הקוד. אופן המימוש של המתודות **search1** ו **search2** אינו רלבנטי לשאלה זו.

```
class SearchService {
  private Map<Character,File> _indexFiles;
  ...
  public List<String> search(String word) {
    try {
      List<Byte> bytes = new LinkedList<Byte>();
      File index = _indexFiles.get(word.charAt(0));
      FileInputStream in = new FileInputStream(index);
     int b = -1;
      while ((b = in.read()) != -1)
        bytes.add((byte)b);
      return search1(word, bytes);
    } catch (Exception e) {
      return new LinkedList<String>();
    }
  }

  protected List<String> search1(String word, List<Byte> bytes) {
    // finds the given word in the given list of bytes, and returns the data (=a list of file names which contains
    // this  word) attached to this word
    ...
  }

  protected List<String> search2(String word, List<ByteBuffer> bytes) {
   // finds the given word in the given list of ByteBuffers, and returns the data (=a list of file names which contains
   //  this  word) attached to this word
    ...
  }
}
```

ב. השתמשו במחלקה SearchService ועדכנו את השרת הנוכחי (הניתן בקוד ה Reactor בנספח), כך שיקבל מהלקוח הודעה המכילה מילה לחיפוש ויחזיר ללקוח את רשימת הקבצים בהם היא מופיעה [6 נקודות]

ג. תארו תרחיש שבו הת'רדים ב Executor נכנסים למצב blocked במהלך ביצוע המתודה processMessage [4 נקודות]

ד. עדכנו את המחלקה SearchService כך שימנע תרחיש ה blocked שתואר בסעיף הקודם (ג). [10 נקודות]

<u>חומר עזר</u>:

- במחלקה FileInputStream קיימת מתודה getFileChannel() המחזירה ייצוג של ערוץ הקלט כ FileChannel.

FileChannel	**getChannel**()

	Returns the unique FileChannel object associated with this file input stream.

- לצורך שאלה זו נניח כי המחלקה **FileChannel** מממשת את הממשק **SelectableChannel** (למרות שזה לא נכון), ובפרט מכילה את המתודה **read**

int	**read**(ByteBuffer dst) Reads a sequence of bytes from this channel into the given buffer.

שאלה 4 (10 נקודות)

בעקבות התרחבות קטלוג הסרטים של מועדון וְשֶׁרֶט לָנֶפֶש, הוחלט לשמור את המידע עבורו בבסיס נתונים רלציוני.

א. הגדירו מודל נתונים עבור קטלוג הסרטים של המועדון המכיל את המידע הבא:
סרטים: שם הסרט, ארץ ושנת הוצאה, שם הבמאי
במאים: שם הבמאי, רשימת סרטים
שחקנים: שם השחקן, רשימת תפקידים בסרטים
[5 נקודות]

ב. כתבו שאילתת SQL המחזירה רשימה של: שם-סרט שם-במאי שחקן תפקיד [5 נקודות]

נספח: החלק העיקרי של קוד ה Reactor, כפי שנלמד בכיתה ובתרגול

```
public class Reactor implements Runnable {
   private static final Logger logger = Logger.getLogger("edu.spl.reactor");
   private final int _port;
   private final int _poolSize;
   private final ServerProtocolFactory _protocolFactory;
   private final TokenizerFactory _tokenizerFactory;
   private volatile boolean _shouldRun = true;
   private ReactorData _data;

   public Reactor(int port, int poolSize, ServerProtocolFactory protocol, TokenizerFactory tokenizer) {
      _port = port;
      _poolSize = poolSize;
      _protocolFactory = protocol;
      _tokenizerFactory = tokenizer;
   }
  ...
   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(executor, selector, _protocolFactory, _tokenizerFactory);

      ConnectionAcceptor connectionAcceptor = new ConnectionAcceptor( 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 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 acceptor = (ConnectionAcceptor) selKey.attachment();
                try {
                    acceptor.accept();
                } catch (IOException e) {
                    logger.info("problem accepting a new connection: "
                            + e.getMessage());
                }
                continue;
            }
            if (selKey.isValid() && selKey.isReadable()) {
                ConnectionHandler handler = (ConnectionHandler) selKey.attachment();
                logger.info("Channel is ready for reading");
                handler.read();
            }
            if (selKey.isValid() && selKey.isWritable()) {
                ConnectionHandler handler = (ConnectionHandler) selKey.attachment();
                logger.info("Channel is ready for writing");
                handler.write();
            }
        } // All fired events have been processed
    } // Either server asked to shutdown the reactor or selector "broke" on an exception.
    stopReactor();
}
…
public static void main(String args[]) {
    if (args.length != 2) {
        System.err.println("Usage: java Reactor <port> <pool_size>");
        System.exit(1);
    }
    try {
                ServerProtocolFactory protocolMaker = new ServerProtocolFactory() {
                    public AsyncServerProtocol create() {
                        return new EchoProtocol();
                    }
                };
```

```
                final Charset charset = Charset.forName("UTF-8");
                TokenizerFactory tokenizerMaker = new TokenizerFactory() {
                    public StringMessageTokenizer create() {
                        return new FixedSeparatorMessageTokenizer("\n", charset);
                    }
                };
            int port = Integer.parseInt(args[0]);
            int poolSize = Integer.parseInt(args[1]);
            Reactor reactor = new Reactor(port, poolSize, protocolMaker, tokenizerMaker);
            Thread thread = new Thread(reactor);
            thread.start();
            logger.info("Reactor is ready on port " + reactor.getPort());
            thread.join();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
public class ConnectionAcceptor {
    private static final Logger logger = Logger.getLogger("edu.spl.reactor");
    protected ServerSocketChannel _ssChannel;
    protected ReactorData _data;

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

    public void accept() throws IOException {
        SocketChannel sChannel = _ssChannel.accept();
        if (sChannel != null) {
            SocketAddress address = sChannel.socket().getRemoteSocketAddress();

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

            ConnectionHandler handler = ConnectionHandler.create(sChannel, _data, key);
            handler.switchToReadOnlyMode(); // set the handler to read only mode
        }
    }
}
```

```
public class ConnectionHandler {
```

```
protected final SocketChannel _sChannel;
protected final ReactorData _data;
protected final AsyncServerProtocol _protocol;
protected final StringMessageTokenizer _tokenizer;
protected Vector<ByteBuffer> _outData;
protected final SelectionKey _skey;
private ProtocolTask _task;
…
public synchronized void addOutData(ByteBuffer buf) {
  _outData.add(buf);
  switchToReadWriteMode();
}

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) {      // Is the channel closed?
    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) { // if nothing left in the output string, go back to read mode
    switchToReadOnlyMode();
    return;
  }
  ByteBuffer buf = _outData.remove(0);
  if (buf.remaining() != 0) {
    _sChannel.write(buf);
```

```
        if (buf.remaining() != 0)
          _outData.add(0, buf);
      }
      if (_protocol.shouldClose()) {
        switchToWriteOnlyMode();
        if (buf.remaining() == 0) {
          logger.info("disconnecting client on " + _sChannel.socket().getRemoteSocketAddress());
          closeConnection();
        }
      }
    }
...
}
```

```
public class ProtocolTask implements Runnable {
    private final ServerProtocol _protocol;
    private final StringMessageTokenizer _tokenizer;
    private final ConnectionHandler _handler;
    private final Vector<ByteBuffer> _buffers = new Vector<ByteBuffer>();

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

    public synchronized void run() {
          synchronized (_buffers) {
              while(_buffers.size() > 0) {
                  ByteBuffer buf = _buffers.remove(0);
                  this._tokenizer.addBytes(buf);
              }
          }
          while (_tokenizer.hasMessage()) {
              String msg = _tokenizer.nextMessage();
              String 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) {
            // We synchronize on _buffers and not on "this" because
            // run() is synchronized on "this", and it might take a long time
            // to run.
            synchronized (_buffers) {
        _buffers.add(b);
            }
    }
}
```

```
public class EchoProtocol implements AsyncServerProtocol {

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

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

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

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

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