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: 8.2.2016
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: א'
משך הבחינה: שלש שעות
חומר עזר: אסור

שאלה 1 (30 נקודות)

הממשק Queue מגדיר תור (FIFO) של אובייקטים.

```
interface Queue<T> {

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

    /** Removes the head item of this queue. Throws NoSuchElementException if the queue is empty. **/
    void remove();

    /** Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. **/
    T peek();

    /** Returns true iff this queue contains no elements. **/
    boolean isEmpty();
}
```

א. הגדירו תכונה נשמרת לממשק ותנאי התחלה וסיום למתודות. אם נדרש, הוסיפו מתודות למחלקה [8 נקודות]

ב. השלימו את מתודות הבדיקה testAdd ו testRemove במחלקה TestQueue (המתודה האבסטרקטית createQueue מחזירה מופע של אובייקט לבדיקה הממשמש את המחלקה, ניתן להניח כי היא ממומשת). [6 נקודות]

```
public class TestQueue {
        private Queue<String> queue;

        protected abstract Queue<String> createQueue();

        @Before
        public void setUp() throws Exception {
           queue = createQueue());
        }

        @After
        public void tearDown() throws Exception {}

        @Test
        public void testAdd() {
           //@TODO
        }

        @Test
        public void testRemove () throws Exception {
           //@TODO
        }
 }
```

להלן הקוד של בעיית הפילוסופים הרעבים שנלמדה בכיתה
[הקוד תואם פחות או יותר את הקוד שנלמד – בוצעו כמה שינויים לא משמעותיים כדי לפשט את הקוד למבחן.
שימו לב כי נעשה שימוש במחלקה Semaphore אותה מימשנו בעצמנו (ללא המתודה tryAcquire, שאינה נדרשת לשאלה זו)].

```
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 Philosopher 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() {
             // the 'thinking' is simulated by a busy wait.
        for (int I = 0; I < 100000; i++);
    }


    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 HungrayPhilosophers {

  public static void main(String [] args){

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

    Philosopher philo1 = new Philosopher(1,fork1,fork2);
    Philosopher philo2 = new Philosopher(2,fork2,fork3);
    Philosopher philo3 = new Philosopher(3,fork3,fork1);

    new Thread(philo1).start();
    new Thread(philo2).start();
    new Thread(philo3).start();
  }
}
```

ג. ציינו האם קיימת סכנת deadlock ו livelock בהרצת התוכנית **HungrayPhilosophers** (כן/לא - תשובה עם הסבר תיפסל), והסבירו מדוע ייתכן starvation.
[6 נקודות]

ד. נתון כי המחלקה **ConcurrentLinkedQueue<T>**, בספריית **concurrent** ב **Java**, מממשת באופן בטוח ונכון את הממשק **Queue<T>**, ללא שימוש בסנכרון (בטכניקה ה lock-free שהודגמה בתרגול על **incrementAndGet**).
פתרו את בעיית ההרעבה בקוד של הפילוסופים הרעבים, על ידי מימוש **Semaphore** הוגן.
[10 נקודות]

שאלה 2 (30 נקודות)

בשאלה זו נדון במנוע משחקי המחשב - SPL3D. במנוע אוסף של צורות (SHAPES) אשר מצוירות על המסך על ידי ציירים (SHADERS). המימוש שלהלן מטפל בצורות תלת ממדיות מסוג BOX ו SPHERE וציירים BASIC ו COMPLEX. ציור צורה למשל BOX על ידי צייר BASIC נעשה על ידי קריאה לפונקציית Render(BOX *b) שלו.

```
class SHADER {…};
class SHAPE {…};
class SPHERE: public SHAPE {…};
class BOX: public SHAPE {…};
class BASIC_SHADER : public SHADER {
        …
        void Render (SPHERE *s) { /* render impl */; }
        void Render (BOX *b)    { /* render impl */; }
};
class COMPLEX_SHADER : public SHADER {
        …
        void Render (SPHERE *s) { /* render impl */; }
        void Render (BOX *b)    { /* render impl */; }
};
class RENDER_MGR {
public:
        RENDER_MGR (int n);
        void Render();
private:
        SHADER **_r;
        SHAPE **_s;
        int     _N;
};
void RENDER_MGR::Render() {
        for (int I = 0; I < _N; i++)
                _r[i]->Render(_s[i]);
}
RENDER_MGR::RENDER_MGR (int n) {
        /*allocates and creates n shaders and shapes*/
}
void main() {
        RENDER_MGR mgr(2);
        mgr.Render();
}
```

א. יש לממש את תבנית ה VISITOR (double dispatch) כך שלולאת הציור RENDER_MGR::Render() תעבוד.
הניחו כי RENDER_MGR תקין לחלוטין, וכי פונקציות הציור במחלקות ה SHADERS כבר ממומשות כפי שמצוין בקוד.
יש להוסיף קוד במחלקות: SHAPE SPHERE, BOX, COMPLEX_SHADER, BASIC_SHADER, SHADER בלבד .
נדרש כי המחלקות SHAPE ו SHADER יהיו PURE ABSTRACT.
[20 נקודות]

ב. ממשו בנאי מעתיק עבור המחלקה RENDER_MGR. יש לבצע העתקה עמוקה (Deep copy).
[10 נקודות]

שאלה 3 (30 נקודות)

כזכור, מימשנו בכיתה שרת הדפסה על בסיס ממשק RMI:

```
public interface Printer extends java.rmi.Remote {
   public void print(String line) throws java.rmi.RemoteException;
 }

public class PrinterImpl  extends java.rmi.server.UnicastRemoteObject  implements Printer  {
 public PrinterImpl() throws java.rmi.RemoteException { }

 public void print(String s)  throws java.rmi.RemoteException {
   System.out.println(s);
 }
}
```

```
public class PrintServer {
 public static void main(String[] args) throws Exception  {
    Printer printer = new PrinterImpl();
    Naming.rebind( args[0] + ":" + Integer.parseInt(args[1] + "/Printer", printer );
   }
}
```

```
public class PrintClient
{
 public static void main(String[] args) throws Exception {
     Printer printerStub = (Printer)Naming.lookup(args[0] + ":" + Integer.parseInt(args[1]) + "/Printer");
     printerStub.print( "hello world" );
 }
}
```

סטודנטית בקורס החליטה לכתוב בעצמה את מחלקות ה Skel וה Stub של **PrinterImpl**, במקום המחלקות הנוצרות אוטומטית על ידי הקומפיילר של RMI.

להלן המימוש שלה למחלקות אלו (הקוד של **MultipleClientProtocolServer**, כפי שנלמד בתרגול, ניתן בנספח למבחן):

```
class PrinterProtocol implements ServerProtocol {

   Printer _printer;

   public PrinterProtocol (Printer printer) {
      _printer = printer;
   }

   public String processMessage(String msg) {
      _printer.print(msg);
   }

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

```
public class Printer_Skel {
   Printer_Skel(Printer printer) throws Exception {
      ServerProtocolFactory  protocolFactory = new ServerProtocolFactory() {
         public ServerProtocol create() {
            return new PrinterProtocol(printer);
         }
      };
      MultipleClientProtocolServer server = new MultipleClientProtocolServer(1984, protocolFactory);
      Thread serverThread = new Thread(server);
       serverThread.start();
   }
}
```

```
public class Printer_Stub implements Printer {
   String  _skelHost;
   int _skelPort;

   Printer_Stub(String skelHost,int skelPort) throws RemoteException {
      _skelHost = skelHost; _skelPort = skelPort;
   }
```

```
  public void print(String str) throws RemoteException {
    try {
      Socket socket = new Socket(_skelHost,_skelPort);
      String msg = str + "\n";
      socket.getOutputStream().write(msg.getBytes("UTF-8"));
      msg = "bye\n";
      socket.getOutputStream().write(msg.getBytes("UTF-8"));
      socket.close();
    } catch (Exception e) {
      throw new RemoteException(e.toString());
    }
  }
}
```

כל אחד מהסעיפים הבאים מתייחס לקוד המקורי הנ"ל, כך שניתן לענות על סעיף למרות שלא עניתם על סעיפים אחרים. בכל מקום בו התבקשתם לעדכן את הקוד, ניתן גם לשנות את הקוד בנספח במידת הצורך.

א. עדכנו את הקוד כך שירוצו בשרת לכל היותר עשרה ת'רדים (השרת יכול לתמוך כמובן ביותר מעשרה לקוחות) [6 נקודות]

ב. עדכנו את הקוד כך שגודל ההודעה – מבחינת מספר בייטים – ניתן בתחילתה (כלומר שההודעה אינה מוגדרת ע"י תו מפריד, כמו סוף שורה). ניתן להניח שגודל ההודעה אינו גדול מ Integer.MAX_VALUE [6 נקודות]

ג. עדכנו את הקוד כך שהתקשורת בין השרת ללקוח נעשית בפרוטוקול UDP. [6 נקודות]

ד. כיתבו SKEL ו STUB, ברוח המימוש של הסטודנטית המצטיינת, עבור ממשק ה Remote הבא [6 נקודות]

```
public interface Math extends java.rmi.Remote {
  // Receives a list of numbers and returns a list of their root numbers.
  List<Float> sqrt(List<Integer> numbers) throws java.rmi.RemoteException;
}
```

ה. עדכנו את הקוד כך שהלקוחות בשרת יטופלו ע"פ תבנית ה Reactor. [6 נקודות]

אם אתם לא זוכרים חתימה כלשהי, תחליטו ע"פ שיקול דעתכם אילו פרמטרים כנראה נדרשים. להלן כמה הגדרות לדוגמא (ניתן להניח כמובן שכל הקוד ממומש כפי שנלמד בכיתה ובתרגול, כמו לדוגמא FixedSeparatorMessageTokenizer).

```
public interface ServerProtocolFactory<T> {
  AsyncServerProtocol<T> create();
}
```

```
interface AsyncServerProtocol<T> extends ServerProtocol<T> {
    boolean shouldClose();
    public void connectionTerminated();
}
```

```
interface TokenizerFactory<T> {
   MessageTokenizer<T> create();
}

interface MessageTokenizer<T> {
   void addBytes(ByteBuffer bytes);
   boolean hasMessage();
   T nextMessage();
   ByteBuffer getBytesForMessage(T msg) throws CharacterCodingException;
}

class FixedSeparatorMessageTokenizer implements MessageTokenizer<StringMessage> { …}
```

שאלה 4 (10 נקודות)

באולפני Abbey Road בלונדון (אולפני ההקלטות המפורסמים של חברת EMI, בהם הוקלטו בין השאר אלבומי הביטלס ופינק פלויד), הצטברו במשך כמויות עצומות של סרטי הקלטה שונים. חברת EMI החליטה לארגן את המידע בארכיון ההקלטות בבסיס נתונים רלציוני, בו יישמרו נתוני ההקלטות.

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.

א. הגדירו מודל נתונים עבור ארכיון ההקלטות. [5 נקודות]

ב. כתבו שאילתת SQL המחזירה את מספרי הטייקים והשירים בהם השתתף בילי פרסטון כקלידן בהקלטות החיפושיות בעונה ההקלטות של הפרויקט get back (ינואר 1969, למי ששכח), ממוינים ע"פ התאריך בסדר עולה [5 נקודות]

נספח: קוד ה MultipleClientProtocolServer, כפי שנלמד בתרגול

```
interface ServerProtocol {

    String processMessage(String msg);

    boolean isEnd(String msg);

}

interface ServerProtocolFactory {
   ServerProtocol create();
}
```

```
class ConnectionHandler implements Runnable {

    private BufferedReader in;
    private PrintWriter out;
    Socket clientSocket;
    ServerProtocol protocol;

    public ConnectionHandler(Socket acceptedSocket, ServerProtocol p) {
        in = null;
        out = null;
        clientSocket = acceptedSocket;
        protocol = p;
        System.out.println("Accepted connection from client!");
        System.out.println("The client is from: " + acceptedSocket.getInetAddress() + ":" +acceptedSocket.getPort());
    }

    public void run()   {

        String msg;

        try {
            initialize();
        }  catch (IOException e) {
            System.out.println("Error in initializing I/O");
        }

        try {
            process();
        } catch (IOException e) {
            System.out.println("Error in I/O");
        }
```

```
        System.out.println("Connection closed - bye bye...");
        close();

    }

    public void process() throws IOException {
        String msg;

        while ((msg = in.readLine()) != null)   {
            System.out.println("Received \"" + msg + "\" from client");

            String response = protocol.processMessage(msg);
            if (response != null)
                out.println(response);

            if (protocol.isEnd(msg))
                break;
        }
    }

    // Starts listening
    public void initialize() throws IOException
    {
        // Initialize I/O
        in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream(),"UTF-8"));
        out = new PrintWriter(new OutputStreamWriter(clientSocket.getOutputStream(),"UTF-8"), true);
        System.out.println("I/O initialized");
    }

    // Closes the connection
    public void close() {
        try {
            if (in != null)
                in.close();
            if (out != null)
                out.close();

            clientSocket.close();
        }
        catch (IOException e) {
            System.out.println("Exception in closing I/O");
        }
    }
}
```

```
class MultipleClientProtocolServer implements Runnable {
    private ServerSocket serverSocket;
    private int listenPort;
    private ServerProtocolFactory factory;


    public MultipleClientProtocolServer(int port, ServerProtocolFactory p)    {
        serverSocket = null;
        listenPort = port;
        factory = p;
    }

    public void run()    {
        try {
            serverSocket = new ServerSocket(listenPort);
            System.out.println("Listening...");
        }
        catch (IOException e) {
            System.out.println("Cannot listen on port " + listenPort);
        }

        while (true) {
            try {
                ConnectionHandler newConnection = new ConnectionHandler(serverSocket.accept(), factory.create());
               new Thread(newConnection).start();
            } catch (IOException e) {
                System.out.println("Failed to accept on port " + listenPort);
            }
        }
    }

    // Closes the connection
    public void close() throws IOException {
        serverSocket.close();
    }
}
```