אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

רשמו תשובותיכם בשאלון זה בלבד
ובמקום המוקצה לכך בלבד !
תשובות מחוץ לשאלון לא יבדקו.

תאריך הבחינה: 6.5.2008
שם המורה: ד"ר מיכאל אלחדד
ניר צחר
אוריאל ברגיג
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשס"ח
סמסטר: א'
מועד: א'
משך הבחינה: שלש שעות
חומר עזר: אסור

בהצלחה!

שאלה 1	(	15 נקודות)

נתון הקוד הבא העוסק בסינכרון ב java ובהמשך 2 סעיפים המתייחסים אליו.

```
public class q1 {
	public static boolean turn = true;
	public static void main(String[] args) {
		Runnable r1 = new Runnable(){
			public void run () {
				while (true) {
					while (turn) {
						System.out.print("a");
						turn = false;
					}
				}
			}
		};

		Runnable r2 = new Runnable() {
			public void run () {
				while (true) {
					while (!turn) {
						System.out.print("b");
						turn = true;
					}
				}
			}
		};

		Thread t1 = new Thread(r1);
		Thread t2 = new Thread(r2);
		t1.start();
		t2.start();
	}
}
```

שאלה 1 - המשך

א. בהנחה שהשמה וקריאה ממשתנה בוליאני מתבצעת ע"י פקודת bytecode ופקודת מכונה אחת, יוסי טוען שהקוד הנתון תמיד מדפיס ...abababababa. הסבר ליוסי מדוע הוא טועה, נמק ע"י דוגמא.

תשובה: התשובה הנכונה הינה שיש בעיית visibility: t1 ירוץ, ידפיס a, וישנה את turn ל-false . אבל, מודל הזיכרון של JAVA אינו מבטיח כי t2 יראה את השינוי, ולכן יתכן מצב שבו לא יודפס יותר שום דבר.
בכל מיקרה, לא יתכן מצב שבו המחרוזת המודפסת אינה רצף כלשהו של abababa.

טעויות נפוצות: אין כאן בעיית reordering.
(Reordering does not happen when function calls and system calls are involved.)

ב. הצע פתרון כך שתמיד יודפס ...abababababa ללא שימוש ב synchronized בקטע הנתון. עם זאת מותר להחליף את המשתנה turn באובייקט ובו מותר להשתמש ב synchronized. עליך לכתוב את המחלקה כולה מחדש.

תשובה: שימו לב, כי לא ניתן להשתמש בסנכרון הקטע קוד הנתון. פתרון אפשרי, הנמצא למטה, משתמש באוביקט בעל שיטות מסונכרנות כדי להבטיח visibility.

```
class Turn{
	private boolean _b = true;
	public synchronized boolean get(){
		return _b;
	}
	public synchronized void flip(){
		_b = !_b;
	}
}
public class q1 {
	public static Turn turn = new Turn();
	public static void main(String[] args) {
		Runnable r1 = new Runnable(){
			public void run () {
				while (true) {
					while (turn.get()) {
						System.out.print("a");
						turn.flip();
					}
				}
			}
		};
		Runnable r2 = new Runnable() {
			public void run () {
				while (true) {
					while (!turn.get()) {
						System.out.print("b");
						turn.flip();
					}
				}
			}
		};
		Thread t1 = new Thread(r1);
		Thread t2 = new Thread(r2);
		t1.start();
		t2.start();
	}
}
```

שאלה 2	)	15 נקודות)

נתון המימוש הבא של תור המאפשר המתנה מרובה על שינויים בו:

```
public interface Notifiable extends Runnable{
	public void callMe();
}

public class Listener implements Notifiable {
	private NotifyingQueue _q1; private NotifyingQueue _q2;
	public Listener(NotifyingQueue q1, NotifyingQueue q2) {
		_q1 = q1; _q2 = q2;
	}
	public void run() {
		_q1.addListener(this);
		_q2.addListener(this);
		while (true){
			try {
				synchronized(this){ this.wait(); }
			} catch (InterruptedException e){}
			System.out.println("been notified");
		}
	}
	public synchronized void callMe(){
		this.notifyAll();
	}
}

public class NotifyingQueue {
	private Vector<String> _queue = new Vector<String>();
	private Vector<Notifiable> _listeners = new Vector<Notifiable>();

	public void addListener(Notifiable n){
		_listeners.add(n);
	}

	public void notifyChange() {
		for (Notifiable n: _listeners)
			n.callMe();
	}

	public void enqueue(String s){
		_queue.add(s);
		notifyChange();
	}

	public String dequeue(){
		if (_queue.size()==0) return null;
		String res = _queue.remove();
		notifyChange();
		return res;
	}

	public static void main (String[] args){
		final NotifyingQueue q1 = new NotifyingQueue();
		final NotifyingQueue q2 = new NotifyingQueue();
		// A thread with multiple wait on q1 and q2
		new Thread(new Listener(q1, q2)).start();
		Thread.currentThread().sleep(20000);
		new Thread(){
			public void run() {
				for (int i=0;i<20;++i){ q1.enqueue("1")}; }
		}.start();
		new Thread(){
			public void run() {
				for (int i=0;i<20;++i){ q2.enqueue("2")}; }
		}.start();
	}
}
```

שאלה 2 - המשך

יוסי שהריץ את הקוד לא הבין מדוע המחורזת been notified מודפסת פחות פעמים מאשר צפוי.

א. [5 נק] - הסבר בקצרה מהי הבעיה והצע פתרון.

תשובה: יש בקוד שתי בעיות:

1. כאשר מתבצע callMe, אין הבטחה שה-thread שמריץ את run של ה-Listener נמצא ב-wait ולכן נפספס notifications
2. אין הבטחה שה-thread שמריץ את run של ה-Listener יספיק לרשום את עצמו בתורים.

פתרון: לספור, באופן בטוח, את מספר הפעמים שנקראה הפונקציה callMe .

ב. [10 נק] – ממש את הפתרון שהצעת. יש לשנות רק את המחלקה Listener

```
public static void main (String[] args) {
        final NotifyingQueue q1 = new NotifyingQueue();
        final NotifyingQueue q2 = new NotifyingQueue();

        // A thread with multiple wait on q1 and q2
        Listener listener = new Listener(q1, q2);
        listener.init();
        new Thread(listener).start();
        new Thread() {
            public void run() {
                for (int i=0;i<20;++i){ q1.enqueue("1");} }
        }.start();
        new Thread() {
            public void run() {
                for (int i=0;i<20;++i){ q2.enqueue("2");} }
        }.start();
    }
}

class Listener implements Notifiable {
    NotifyingQueue _q1, _q2;
    Semaphore _sem = new Semaphore(0);
    public Listener(NotifyingQueue q1, NotifyingQueue q2) {
        _q1 = q1; _q2 = q2;
    }
    public void init(){
        _q1.addListener(this);
        _q2.addListener(this);
    }
    public void run() {
        while (true){
            boolean acquired = false;
            while (!acquired) {
                try{
                    _sem.acquire();
                    acquired = true;
                } catch (InterruptedException ignored) {}
```

```
                }
                System.out.println("been notified");
            }
        }
        public void callMe() {
            boolean released = false;
            while (!released) {
                try {
                    _sem.release();
                    released = true;
                } catch (InterruptedException ignored) {}
            }
        }
}
```

בעיות נפוצות:

1. העברת System.out.println("been notified") לתוך callMe: מי שידפיס את השורה הינו ה-thread שביצע את עדכון התורים, ולא ה-thread שאמור לקבל את הנוטיפיקציה. תשובה מסוג זה קיבלה את מחצית הניקוד.
2. עטיפת ה-while ב-synchronized(this) : הפתרון אינו נכון, מכיוון שעדיין ניתן לפספס נוטיפיקציות.
3. שימוש ב-Sleep בוריאציה כלשהי: פתרון זה אינו נכון. אף פעם אין להשתמש ב-sleep על מנת לפתור בעיות סינכרון.
4. כאשר ה-Listener נמצא ב-wait() , אזי המנעול שלו נעול ואי אפשר להפעיל את המתודה callMe: תשובה זו אינה נכונה, מכיוון שכאשר הוא נכנס ל-wait , הוא ויתר על המנעול.

שאלה 3	(	16 נקודות. 4 לכל סעיף)

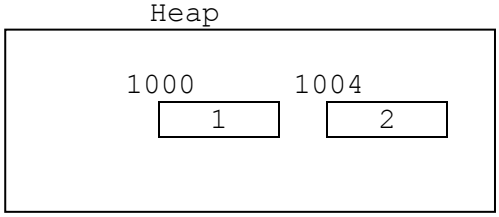
בשאלה זו נתונים קטעי קוד ב ++C. לכל קטע קוד עליכם לצייר את תמונת הזכרון של ה- Stack וה- Heap כפי שהיא נראת במקומות המסומנים בחץ. להלן דוגמא:

```
{
      int i=3;
      int *p1 = new int(1);
      int *p2 = new int(2);
                                  ←———— תמונת הזכרון כאן:
}
```

Stack

884	
888	
892	1004
896	1000
900	3

Heap



SP = 892

שימו לב:

א)ניתן להניח כי כל נתון מכל סוג גודלו 4.

ב) בכל המקרים, ה-Stack Pointer מכיל לפני ביצוע הקוד את הערך 904.

ג) יש לציין את מיקום ה Stack Pointer במחסנית.

ד) יש להצמד לצורת השרטוט שבדוגמא גם אם היא אינה מייצגת בדיוק את סידור הזכרון הידוע לכם. הסברים במילים אינם מתקבלים.

```
class Q
{
private:
      int i;
      int *pi;
      int *p;
public:
      Q(int k);
      print();
};
------------------
Q::Q(int k)
{
      i = k;
      pi = &i;
      p = new int(3);
}

Q::print()
{
      cout << "i=" << i << " pi="<< pi << " p=" << p;
}

main()
{
      Q q(2);
      q.print();
                  ←———— סעיף א - תמונת הזכרון כאן:
}

  ←———— סעיף ב - תמונת הזכרון כאן:
```

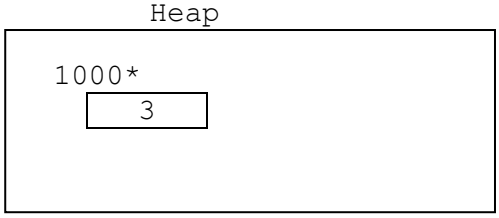
פתרון סעיף א

Stack

880	
884	
888	
892	1000*
896	900
900	2

SP = 892

Heap



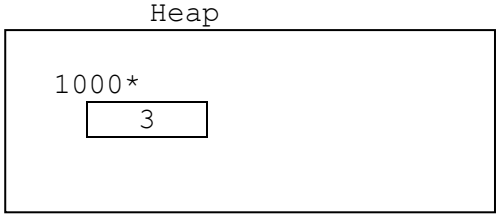
פתרון סעיף ב

Stack

880	
884	
888	
892	
896	
900	

SP = 904

Heap



הערות:

- * מציינת מספר שרורתי
- בסעיף ב' ניתן להשאיר את הערכים על המחסנית העיקר שה- SP תקין

נוסיף destructor בהגדרה של Q. להלן המחלקה Q כתובה מחדש

```
class Q
{
private:
      int i;
      int *pi;
      int *p;
public:
      Q(int k);
      print();
      ~Q() {delete pi; delete p;}
};
```

נוסיף מחלקה המרחיבה את Q:

```
class DQ : public Q
{
private:
      int *pd;
public:
      DQ();
      ~DQ();
};
-------------
DQ::DQ():Q(2)
{
      pd = new int(1);
}
DQ::~DQ()
{
      delete pd;
}
```

שאלה 3 - המשך

```
main()
{
      int i = 5;
      Q *p = new DQ();
      p->print();
      delete p;
```

סעיף ג - תמונת הזכרון כאן:

```
}
```

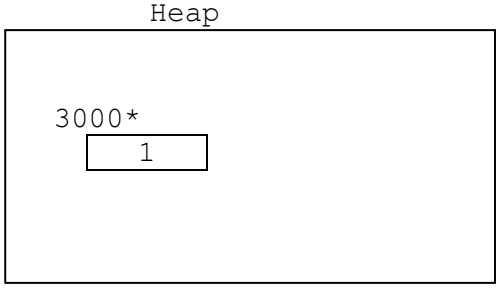
סעיף ד - תמונת הזכרון כאן:

פתרון סעיף ג

Stack

880	
884	
888	
892	
896	1000*
900	5

Heap



הערות:

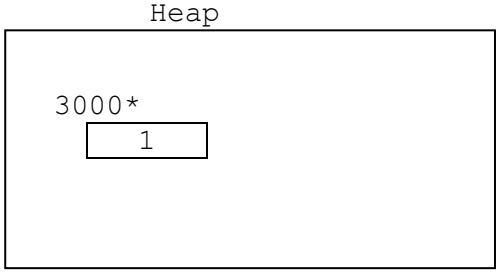
- טעויות נפוצות:
  - רשימת הערכים של האובייקט DQ על המחסנית והוספת ערכים לא קשורים למחסנית
  - אי זיהוי דליפת זכרון או זיהוי שגוי שלה: אי שימוש ב virtual גורם לקריאה לפונקציה לפי הסוג הידוע בזמן קומפילציה קרי: Q. כלומר הדיסטקרטור שלא יקרה בוודאות הוא זה של DQ
- כל הנחה לגבי Q~ מותרת ולכן ניתן היה להשאיר את הערך 3 על ה heap

פתרון סעיף ד

Stack

880	
884	
888	
892	
896	
900	

Heap



הערות:

חשוב היה (1) לזהות שיש דליפה. (2) לזהות אותה במדוייק.
ניתן היה להשאיר ערכים על ה stack.

כיון שהנתונים נוספים על ה heap אמורים היו להיות מחוקים הם לא נבדקו, עם זאת במקרים רבים נדמה היה שחלק מהתשובות אינן משקפות הבנה של כיצד נראה הזכרון של אובייקט שיורש מאובייקט אחר.

שאלה 4	(	14 נקודות)

נתונה המחלקה הבאה:

```
class IntData
{
private:
      const int _i;
public:
      IntData(int i);
      void printData() const;
      virtual ~IntData();
};
----------------
IntData::IntData(int i):_i(i){}
void IntData::printData()const
{
      cout << "IntData = " << _i << endl;
}
IntData::~IntData()
{
      cout << "deleting data " << _i << endl;
}
```

כמו כן נתונה הגדרתה של המחלקה הבאה:

```
class SmartPointer
{
      IntData *_p;
      int *_refCount;
public:
      SmartPointer(IntData *p);
      SmartPointer(const SmartPointer& o);
      IntData *getData() const;
      virtual ~SmartPointer();
};
```

ונתון main המשתמש במחלקות אלו:

```
main()
{
      SmartPointer sp1(new IntData(1));
      IntData *p1 = sp1.getData();
      cout << p1  << endl;
      p1->printData();
      SmartPointer *sp2 = new SmartPointer(sp1);
      IntData *p2 = sp2->getData();
      cout << p2  << endl;
      p2->printData();
      delete sp2;
      cout << "AFTER sp2 DELETE" << endl;
}
```

סעיף א' [7 נקודות]

עליכם לממש את המחלקה `SmartPointer` כך שבהרצת ה main יתקבל הפלט הבא:

```
00263450
IntData = 1
00263450
IntData = 1
AFTER sp2 DELETE
deleting data 1
```

שאלה 4 - המשך

פתרון סעיף א'

```
SmartPointer::SmartPointer(IntData *p):
      _p(p), _refCount(new int(1))  {};

SmartPointer::SmartPointer(const SmartPointer& o):
      _p(o._p), _refCount(o._refCount) {
      (*_refCount)++;
}

IntData *SmartPointer::getData() const {
      return _p;
}

SmartPointer::~SmartPointer() {
      (*_refCount)--;
      if (0 == *_refCount) {
            delete _p;
            _p = 0;
            delete _refCount;
            _refCount = 0;
      }
}
```

הערות:

שגיאה נפוצה היתה _refCount(new int(2)) ב copy constructor. ניתן היה להשים את הערך 2 במקום לבצע ++ על אף שהכוונה היתה לממש משהו כללי אך אסור היה להשתמש ב new כי הכוונה היא שהמונה refCount יהיה משותף לכל הפיונטרים שמצביעים לאותו נתון.

סעיף ב' [7 נקודות]

למחלקה SmartPointer נוסף אופרטור ההשמה:

```
class SmartPointer
{
      ...
public:
      ...
      SmartPointer& operator=(const SmartPointer &o);
};
```

שאלה 4 - המשך

להלן main המשתמש באופרטור זה ותאור פלט ההרצה

```
main()
{
        SmartPointer sp1(new IntData(1));
        IntData *p1 = sp1.getData();
        cout << p1  << endl;
        p1->printData();
        SmartPointer sp2(new IntData(2));
        IntData *p2 = sp2.getData();
        cout <<  p2  << endl;
        p2->printData();
        sp1 = sp2;
        IntData *p3 = sp1.getData();
        cout << p3  << endl;
        p3->printData();
}
```

```
4142072
IntData = 1
4144808
IntData = 2
deleting data 1
4144808
IntData = 2
deleting data 2
```

ממש את אופרטור ההשמה

```
SmartPointer& SmartPointer::operator=(const SmartPointer& o)
{
      if ( this == &o ) return *this;

      (*_refCount)--;
      if (0 == *_refCount) {
            delete _p;
            delete _refCount;
      }
      _p = o._p;
      _refCount = o._refCount;

      return *this;
}
```

הערות:

חשוב היה להראות את ארבעת השלבים שבאופטור ההשמה:

self assignment check, clear, copy, return *this

שאלה 5	)	30 נקודות)

התבוננו בקוד השרת של מחלקת EchoProtocol.

```
interface ServerProtocol {
    String processMessage(String msg);
    boolean isEnd(String msg);
}

class EchoProtocol implements ServerProtocol {
    private int counter;
    public EchoProtocol() { this.counter = 0; }
    public String processMessage(String msg) {
        this.counter++;
        if (this.isEnd(msg)) {
            return new String("Ok, bye bye...");
        } else {
            final SAXBuilder builder = new SAXBuilder();
            Document doc;
            try {
                final StringReader stringReader = new StringReader(msg);
                doc=builder.build(stringReader);
                final Element root = doc.getRootElement();
                final String encodedMsg = root.getText();
                final byte[] bytes = Base64.decode(encodedMsg);
                final String decodedMsg = new String(bytes,1,bytes.length-1);
                return new String(this.counter + ". Received \"" + decodedMsg);
            } catch (JDOMException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
            return null;
        }
    }
    public boolean isEnd(String msg) { return msg.equals("bye"); }
}

class ConnectionHandler implements Runnable {
    private BufferedReader in;
    private PrintWriter out=null;
    Socket clientSocket=null;
    ServerProtocol protocol;
    public ConnectionHandler(Socket acceptedSocket, ServerProtocol p) {
        this.clientSocket = acceptedSocket;
        this.protocol = p;
    }
    public void run() {
        try { this.initialize(); } catch (IOException e) { … }
        try { this.process(); }   catch (IOException e) { … }
        System.out.println("Connection closed - bye bye...");
```

```
48.         this.close();
49.     }
50.     public void process() throws IOException {
51.         String msg;
52.         while ((msg = this.in.readLine()) != null) {
53.             final String response = this.protocol.processMessage(msg);
54.             if (response != null) this.out.println(response);
55.             if (this.protocol.isEnd(msg)) break;
56.         }
57.     }
58.     public void initialize() throws IOException {
59.         this.in = new BufferedReader(
60.                         new InputStreamReader(this.clientSocket.getInputStream()));
61.         this.out = new PrintWriter(this.clientSocket.getOutputStream(), true);
62.     }
63.     public void close() {
64.         try {
65.             if (this.in != null)  this.in.close();
66.             if (this.out != null) this.out.close();
67.             this.clientSocket.close();
68.         } catch (IOException e) { … }
69.     }
70. }
71.
72. class EchoProtocolServer implements Runnable {
73.     ServerSocket serverSocket=null;
74.     int listenPort;
75.     public EchoProtocolServer(int port, ServerProtocol p) {
76.         this.listenPort = port;
77.     }
78.     public void run() {
79.         try {
80.             this.serverSocket = new ServerSocket(this.listenPort);
81.             System.out.println("Listening...");
82.         } catch (IOException e) { … }
83.         while (true) {
84.             try {
85.                 ConnectionHandler newConnection = new
86.                 ConnectionHandler(this.serverSocket.accept(), new EchoProtocol());
87.                 new Thread(newConnection).start();
88.             } catch (IOException e) { … }
89.         }
90.     }
91.     public void close() throws IOException {
92.         this.serverSocket.close();
93.     }
```

סעיף א' [10 נקודות]

איזה Framing Protocol השרת מצפה לקבל? רשם את מספרי שורות קוד המטפלות ב-framing.

Answer:

The framing protocol is the part of the protocol that determines how one message is separated from the next message.
(See http://www.cs.bgu.ac.il/~spl081/Protocols for the definition given in class.)

In the code, framing is the part that is "closest" to the part that handles reading from the input socket.

In the code given (which was the code reviewed in Tirgul 11 http://www.cs.bgu.ac.il/~spl081/Practical_SessionsAss3) this was the code written in line 52:

```
while ((msg = this.in.readLine()) != null)
```

The key point is that the reader decides that a message is finished when if reads a line that ends with a newline character. Using the standard library readLine() and the corresponding println() (Line 54) is a very easy way to handle message framing.

Note that framing is different from the serialization format used by the protocol: the serialization format determines what is the syntax of the messages exchanged by the sides of the protocol. The serialization format defines the format of messages "inside the envelope" – while the framing protocol defines the "shape of the envelope".

In the code provided, the serialization format was:

- Either the string "bye" (indicating the end of session)
- Or an XML string formed of a single root element containing a string encoded in Base64. (The decoding of this format is handled in lines 17-22.)

In both cases, the serialized format is "encapsulated" in the framing format by adding a newline at the end of the string.

Frequent mistakes:

- Confuse framing and serialization protocols
- Confuse message delimiters (newline) and end-of-session indicator (bye)

Those who described the serialization protocol instead of the framing protocol got 6 out of 10 points.

סעיף ב' [10 נקודות]

יוסי רוצה להרחיב את שרת XMPP שהוא מימש בעבודה 3 כך שיתמוך בהעברת קבצי תמונות מסוג JPEG. הדרישה היא לתמוך בתרחיש הבא:

```
XMPP Client Yosi: Send file /usr/images/cake.jpg name:cake
XMPP Client Sara: [Yosi sent you a picture called "cake"]
XMPP Client Sara: [To get the picture type: get picture cake]
XMPP Client Sara: get picture cake
XMPP Client Sara:
```

הגדר את פורמט ההודעות הדרושות למימוש הרחבת הפרוטוקול הזו.
על הפרוטוקול להעביר את תוכן התמונה בפורמט JPEG הבינארי.

רשם את כל ההודעות הדרושות ככוון client→server וו server→client.

על ההודעות להיות במבנה דומה לזה של שאר הודעות XMPP כלומר XML של stanza.

Answer:

Several answers were possible – I present here the simplest protocol and discuss variations in the next section.

Background:

The part to remember from Assignment 3 was that the XMPP protocol is about exchanging "XML stanzas" between client and server – in other words, the framing protocol of XMPP is to encapsulate messages in tags of the form <tag>…..</tag>. The specific stanzas that were described and used in Assignment 3 were <message>…</message>, <presence>…</presence> and <iq>….</iq>.

To extend the XMPP protocol, one should define a new stanza tag and specify the structure of its child elements to convey the information requested by the protocol extension.

Pre-requisite clarification:

What is called "binary format" is as opposed to "textual format".
This point was explained in class (http://www.cs.bgu.ac.il/~spl081/Protocols section "Binary Data"). A textual format uses a specific textual encoding (ASCII, UTF8, UTF16 etc) to encode characters. A binary format has its own internal syntax and does not need to use characters from a specific character set. For example, the format of a JPEG file is specified by a standard called the JFIF (cf. http://www.w3.org/Graphics/JPEG/jfif3.pdf or http://en.wikipedia.org/wiki/JPEG_File_Interchange_Format). The important point to remember though, is that ALL computer data is always encoded eventually in "binary" – meaning, all encodings use sequences of bits.

Sending a binary format means that the bits used in the binary format are all sent from one side to the other. It does NOT mean sending a "string of characters that encode each 0 or 1".

Sending binary data is a problem when we need to design the FRAMING part of a protocol. The reason is that, if we decide to use framing that relies on delimiters (newlines or XML tags

for example), then it is difficult to encapsulate arbitrary binary data because the binary data may include the delimiters in places we do not expect. The reason is that in a binary format, we are not restricted to a specific list of characters in the content – any combination of bits can appear anywhere. If we interpret the stream of bits as a sequence of characters, we may end up with the value of the delimiter string "by mistake".

Protocol 1:

The "normal sequence" of messages is:

1. Client A to Server: Submit a file to upload
2. Server to Client A: Acknowledge (OK or Not-OK)
3. Server to Client B: Notify that a file is available for download
4. Client B to Server: Request file download
5. Server to Client B: Deliver file data (or failure)

Now as to possible encodings for the messages:

Client A to Server: Upload file

```
<image:upload from='yosi@spl" to='sara@spl'>
    <image:name>cake</image:name>
    <image:filename>/usr/images/cake.jpg</image:filename>
    <image:data>[jpeg file encoded in Base64]</image:data>
</image:upload>
```

Server reply to client A:

Upload was accepted, notification was sent to target user, image remains available on server so that target user can download it later and is assigned a unique ID for reference:

```
<image:upload type='result'>
    <image:id>cake-1A3X</image:id>
</image:upload>
```

Upload was refused by server:

```
<image:upload type='error'>
    <error>No storage available</error>
</image:upload>
```

Server to Client B:

Since this is a "normal" notification, we can reuse an existing XMPP <message>.
(It was also ok to introduce a new message for this.)

```
<message from='yosi@spl' to='sara@spl'>
    <body>Yosi sent you a picture called 'cake'.  To retrieve it…</body>
</message>
```

Client B to Server: Request file download

```
<image:download>
    <image:name>cake</image:name>
</image:download>
```

Server to Client B: Deliver data

If the server finds the picture (based on its name and requesting user) – deliver the data:

```
<image:download type='result'>
    <image:name>cake</image:name>
    <image:id>cake-1A3X</image:id>
    <image:data>[jpeg file encoded in Base64]</image:data>
</image:download>
```

If the server fails to find the picture or refuses to give access to it – send an error:

```
<image:download type='error'>
    <image:name>cake</image:name>
    <error>Image not available</error>
</image:download>
```

The key points in the grading were:

- Define the right steps in the protocol (Error messages were not expected – but the logic of the 5 steps was expected.
- Propose a reasonable XML encoding for messages.
- Address the issue of encoding the binary format (using base64 or any variation of this addresses the issue of encapsulating binary data in a delimiter-based framing format).

Frequent errors:

- Fail to transfer the content of the file
- Fail to encode the binary data in a way that is safe to the XML framing

Note:
It is natural to use a specific namespace for a protocol extension (I used here image:) to avoid confusion with existing tags used by the "base" protocol.

סעיף ג' [10 נקודות]
דני התלונן שהקידוד להודעות שמכילות תמונות בזבזני.
הסבירו למה.
הציעו שיטת העברת תמונות בפרוטוקול יותר יעילה.

Answer:

Pre-requisite clarification: "wasteful" (בזבזני) does not mean the same thing as "expensive". If the protocol requires us to send a picture from Yosi to Sara – then we must send all the bits of the picture from Yosi to Sara, even if there are many bits and the picture is large. This is an expensive operation – but if it is required, there is no way to "cut corners" – that's what we have to do.

Wasteful is something different: it means we do "extra" operations which are not absolutely necessary, and while such extra operations may be ok when we send "small messages", they may become unbearable when we deal with expensive messages.

What are the wasteful aspects of the protocol described above?

- The file content is encoded in base64. Base64 turns binary data into a sequence of ASCII characters. When doing this, it expands the data by a factor of approximately 1.5 (100 bytes become 150 bytes). For large files, this 50% penalty is wasteful.
- We transmit the file twice – from client A to server, then from server to client B. It could be more efficient to send directly from client A to client B.
- We must store the file on the server until client B decides whether to download it. This storage may be unnecessary, and requires a process on the server side to "clean up" (garbage collect) files after a while. This process requires both storage space and processor time on the server side which is not absolutely necessary.

Possible solutions:

- **Use a different framing mechanism for the file data**. This would mean that for the upload and download messages, we would first send notification that the data is

available, and then communication would continue on a different port using a different framing (size + data) not using delimiters.

- **Allow peer to peer file transfer**. When client A notifies the server that he wants to send a file, do not upload the data to the server – but just notifies client B. Then client B asks the server for direct access to client A (IP address and special port for the file transfer). From that point on, Client B accesses Client A directly and negotiates the file transfer with client A. This requires developing a server process on Client A's side.
- **Upload the file from Client A to Server on demand only**: to avoid the problem of storing the files on the server when they may not be used, proceed as follows: when Client A notifies server that he makes a file available for upload (but the file content is not sent); Server notifies Client B; Only when client B requests the file, the server turns to Client A and requests the file content; Client A sends file content; Server reads the file content and passes it down to Client B without storing it on its side.

Notes:
In most "open network" applications, implementing a "peer to peer" direct communication is difficult. This is because clients run firewalls (special software that blocks most IP ports) and a mechanism called NAT (Network Address Translation http://en.wikipedia.org/wiki/Network_address_translation) .

Full grading was granted if any of the problems above was mentioned and any of the possible solutions was described.

Common errors:
- UDP would not help – the problem is related to the protocol and to the framing; splitting a large packet of data into smaller packets does not make the transfer more efficient.
- RMI does not help – RMI itself is implemented in terms of TCP communication. The problem of serializing a file in RMI is identical to serializing in TCP.
- Passing a "file reference" does not help – we cannot assume Client B has access to the filename in the file system of Client A + even if it were accessible, the bits still need to be transferred from Client A to Client B.

שאלה 6	(	10 נקודות)

בשאלה זו הנכם נדרשים להגדיר מודל נתונים שלם ומינימלי עבור התאור הבא, על מנת לתמוך בשאילתות מהסוג שיפורט בהמשך:
אולפן ההפקות splProductionsLTD שכר אתכם לתכנן את בסיס נתונים חדש. האולפן עוסק בעיקר בהפקת סדרות, ומעוניין לנהל מעקב אחר הסדרות שהופקו באולפן. מנהל האולפן מסר לכם את המידע הבא: לכל סידרה יש שם יחודי. בנוסף, כל סידרה בוימה על ידי במאי יחיד. אולם, בכל סידרה השתתפו מספר שחקנים שונה, אשר כל אחד מהם הרויח סכום כסף שונה עבור משחקו בסדרה. בנוסף, כל סידרה נימכרה למספר תחנות טלויזיה שונות, בסכומי כסף משתנים.

סוגי השאילתות המעניינות את אולפן ההפקות הן: שמות הבמאים שסדרות שהם הפיקו נרשכו יותר ממספר פעמים כלשהו, שמות השחקנים ששיחקו בסדרה מסוימת, כמה כסף עלתה הפקת כל סידרה (מבחינת משכורת השחקנים).

עליכם להציג מודל נתונים מתאים, ולציין מיהם המפתחות הראשיים והזרים.

Solution:

Identification of the key objects and their attributes:
- Series (seriesName varchar(200) PK, directorID int)
- Actor (actorName varchar(200), actorID int PK)
- TVStation (stationName varchar(200), stationID int PK)
- Director (directorName varchar(200), directorID int)

Primary Keys of the objects:
- Series: name is unique (was specified)
- Actor: we cannot assume name is unique, so we'll use an integer ID.
- TVStation: we cannot assume name is unique, so we'll use an integer ID.

Relations among objects:

- The same director can direct several series
- Several actors play in one series
- The same actor can play in several series
- When an actor plays in a series, he is paid a salary
- The same series can be sold to several TV Stations
- One TV Station can buy several series

Hence we derive the need for cross-tables for all n-to-n relations:
- ActorSeries (actorID int, seriesName varchar(200), salary int)
- StationSeries(stationID int, seriesName varchar(200), cost int)

The 2 keys of the cross tables form primary keys:
Primary Key of ActorsSeries (actorId, seriesName)
Primary Key of StationSeries (stationId, seriesName)

And the foreign keys:
- Series.directorID is a foreign key that references Director.directorID
- ActorSeries.actorID references Actor.actorID
- ActorSeries.seriesName references Series.seriesName
- StationSeries.stationID references TVStation.stationID
- StationSeries.seriesName references Series.seriesName