גיליון תשובות

מספר נבחן:_____

Q1	Q2
Q3	Q4
Q5	Q6
TOTAL	

שאלה 1 (30 נקודות)

סעיף א (6 נקודות)

1.1.א

a set to 1
c set to 0
b set to 1
c set to 1

1.2.א

b set to 1
c set to 0
a set to 1
c set to 1

2.א

יש הרבה... בתשובה הספיקו:

a set to 1
c set to 0
b set to 1
c set to 1

b set to 1
c set to 0
a set to 1
c set to 1

a set to 1
b set to 1
deadlock

```
a  set to 1
b  set to 1
deadlock
```

הוחלט לקבל כתשובה נכונה גם תשובות שהתעלמו מה deadlock בשני הפלטים האחרונים כלומר:

```
a  set to 1
b  set to 1
c  set to 0
c  set to 1

a  set to 1
b  set to 1
c  set to 0
c  set to 1
```

סעיף ב (7 נקודות)

קיימות מספר אפשרויות, לדוגמא:

T1

```
wire1.setValue(1);
```

נעילה של wire1 ל T1

```
System.out.println(_id + " set to " + value);
_value = value;
_toGate.onChange();
```

נעילה של gate ל T1

```
_inwire1.getValue();
```

T2

```
wire2.setValue(1);
```

נעילה של wire2 ל T2

```
System.out.println(_id + " set to " + value);
_value = value;
```

T1

```
_inwire2.getValue();
```

מחכה שתשתחרר הנעילה מ wire2 התפוס על ידי T2

T2

```
_toGate.onChange();
```

מחכה שתשתחרר הנעילה מ gate התפוס על ידי T1

סעיף ג (7 נקודות)

האינווריאנטה: הערך בחוט היוצא מהשער הוא החיתוך הלוגי של החוטים הנכנסים.
נדגיש כי מבחינת המודל שלנו (הלוא הוא מודל האובייקטים), ערך של חוט הוא כזה המתקבל על ידי מתודה פומבית, דוגמת getValue(), ולא ערכו מבחינת השדה _value.

```
boolean synchronized public test()
{
  synchronized(_inWIre1)
    synchronized(_inWIre2)
      synchronized(_outWIre)
        return (_outWire.getValue() = =    _outWire.getValue() & _outWire.getValue())
}
```

הגדרת האינווריאנטה כשמירה על ערכי החוטים כ 0 או 1 התקבלה באופן חלקי בלבד.

סעיפים ד-ה (10 נקודות)

נתבונן במחלקה BasicAndGate.
שתי תכונות מרכזיות בולטות מיד לעין:
- המצב הפנימי של המחלקה מורכב מאובייקטים חיצוניים אשר עלולים להיות חלק משערים אחרים או כל אוביקט אחר.
- פעולות בסיסיות על השער מוגדרות מעל גישה למספר אוביקטים.
שתי תכונות אלו ממחישות בצורה טובה שתי בעיות מרכזיות בחישוב: שמירה על אינווריאנטה וחבק.
שימוש מסיבי בסנכרון יגרום קרוב לוודאי לחבק, בעוד שוויתור על הסינכרון יגרום לאובדן הבטיחות.

1. בעיית הבטיחות המרכזית קשורה ביכולת לשנות את החוט היוצא שלא דרך השער, תוך הפרה בוטה של התכונה הנשמרת. בעיה זו אינה ייחודית לחישוב מקבילי.
אחת הדרכים למנוע אותה היא על ידי הגדרת המתודה setValue של wire כך שלא ניתן לבצעה שלא משער המקור:

```
public void synchronized setValue(int value,Object key) {
  if (_sourceKey == key) {
    System.out.println(_id + " set to " + value);
    _value = value;
    _toGate.onChange();
}
```

כאשר key הוא אוביקט פרטי ללא גישה פומבית המיוצר במחלקה BasicAndGate והניתן לחוט בזמן חיבורו לשער (הוא מחובר לשער נכנס אחד בלבד):

```
class BasicAndGate implements Gate
{
        protected Wire _inwire1;
        protected Wire _inwire2;
        protected Wire _outwire;
```

```
        private Object _outwireKey;

        BasicAndGate() { }

public synchronized void connect(Wire inwire1, Wire inwire2, Wire outwire) {
        _inwire1 = inwire1;
        _inwire2 = inwire2;
        _outwire = outwire;
        _outWireKey = new Object();
        _outWire.setSourceKey(_outwireKey);
        onChange();
}

public synchronized void onChange()  {
      if (outwIreKey != null)
          _outWire.setValue(_inwire1.getValue() & _inwire2.getValue(), _outwIreKey);
}
```

תשובות בהם יוצר החוט עצמו בתוך השער לא קבלו את מלוא הנקודות, כי מבחינת המערכת החוטים הם אוביקטים עמצאיים, ניתן לחברם לשער אחר בצדם השני, וכן הלאה.

2. כעת נבחן את הפונקציה ()onChanged של השער. כדי לחשב את ערך החוט היוצא, השער קורא בנפרד את הערך בשני החוטים הנכנסים. לכאורה קיימת בעיית בטיחות שכן קוראים את הערך בחוט הראשון ורק אח"כ את הערך בחוט השני מבלי לסנכרן קודם את שניהם, כך שיתכן שערך אחד החוטים הנכנסים השתנה על ידי ת'רד אחר והערך המחושב על ידי השער לחוט היוצא כבר לא נכון.
ציינו קודם כי התכונה הנשמרת של המחלקה אינה על הערך של החוטים עצמם (כפי שמוגדר בשדה _value) אלא על ערכם מבחינת מודל האובייקטים, כלומר דרך המתוד הפומבי ()getValue. באופן שכזה, על אף שת'רד אחר שינה את ערכו של חוט כניסה לאחר שהת'רד הראשון קרא את ערכו לשם חישובו של החוט היוצא, עדיין המחלקה בטוחה, שכן קריאה ל ()getValue של החוט לא תחזיר את ערכו החדש של החוט אלא תיתקע עד אשר הנעילה של החוט מ ()setValue תשתחרר, וזה יקרה רק אחר הסיום של onChanged על השער עם הערך החדש. לסיכום, בכל מצב לא יציב של המחלקה, הגישה הפומבית לערכי החוטים מתאימה לתכונה הנשמרת של השער ביציאה מ onChanged
אם כדי למנוע deadlock היינו מוציאים את הקריאה ל ()onChenged מהסנכרון של ה wire:

```
public void setValue(int value) {
        synchronized(this) {
                System.out.println(_id + " set to " + value);
                _value = value;
        }
        _toGate.onChange();
}
```

הבעיה הנ"ל הייתה מתרחשת, והיה נדרש התיקון ב ()onChanged של BasicAndGate:

```
public synchronized void onChange()  {
        synchronized(_inwire1)
          synchronized(_inwire2)
                _outWire.setValue(_inwire1.getValue() & _inwire2.getValue());
}
```

או לחלופין סנכרון של _toGate מתוך ()setValue של wire.

- מי שהסביר במפורט למה המחלקה בטוחה (מבחינת 2) קיבל את מלוא הנקודות של סעיפים ד-ה. במידה וההסבר לא פרט כיצד הבעיה האפשרית לא באה לידי ביטוי לאור מודל האובייקטים, הופחתו מספר נקודות.

- הוחלט לתת את מלוא הנקודות, למי שראה בבעיה 2 בעיית בטיחות ותיקן אותה על סמך זאת כנדרש.

3. מאחר שעלול להתרחש חבק, ניתן לראות זאת כחוסר התאמה של החישוב הסדרתי לתוצאת חישוב סדרתי. על אף שהשאלה התמקדה בהגדרת בטיחות כשמירה על אינווריאנטה, ניתן ניקוד, כזה או אחר, על תשובות ממין זה.

4. מאחר שהמשתנים של החוטים הם protected ניתן תאורטית לשנות אותם באופן לא מסונכרן מתוך מחלקה הנורשת מ BasicAndGate. ניתן גם ניקוד (לא בהכרח מלא) למי שעמד על בעיה זו.
(אגב, לא ניתן לגשת ל protected member מתוך "חבילה" אלא רק ממחלקה יורשת).

5. עבור אלו שדבקו בהגדרת האינווריאנטה כשמירה על ערכי 1,0 בחוטים ניתן ניקוד חלקי.

בכל מקרה הניקוד לכל נבחן/נת היה בהתאם למסלול הייחודי בו הוא/היא הלך/כה, ולאור תשובותיו/יה בסעיפים הקודמים, תוך עמידה על חשיבה מקורית ויוצרת.

שאלה 2 (12 נקודות)

סעיף א (6 נקודות)

0
1
פניה למקום לא חוקי בזכרון

הבעיה: לא מוגדר אופרטור השמה, כך שמופעלת ברירת המחדל המעתיקה ביט אחר ביט, כך שהמצביע _p של a2 ו a3 הוא בעל אותו ערך – כלומר מצביע לאותו מקום בזכרון. בסיום ה scope של a3, קרי הבלוק הפנימי, יופעל ה destructor שלו והזכרון אליו מצביע _p ימחק. הפניה של a2 למקום זה היא אם כן לא חוקית.

סעיף ב (6 נקודות)

יש להגדיר את אופרטור ההשמה עבור המחלקה A:

```
class A {
public:
.
.
.
  A& operator=(const A& other) {
     If (this != &other) {
        delete _p;
        _p = new int(*(other._p));
     }
     return *this;
```

```
 }
.
.
.
};
```

שאלה 3 (18 נקודות)

סעיף א (15 נקודות)

1.א 0,2

2.א פניה למקום לא מוגדר במחסנית

3.א 0,2

4.א פניה למקום מחוק ב heap

5.א 0,2

סעיף ב (3 נקודות)

smooh3

או

smooth5

שאלה 4 (17 נקודות)

סעיף א (5 נקודות)

The server is a TCP server. The application protocol includes a single request:

- GetAddressOfPlayer(in string MovieName, out string AddressOfPlayer)
  errors: No such movie currently playing

There are two possible answers based on the assumptions:

<u>Assumptions 1</u>:
- There is a "small" number of players (for example - less than 100).
- The list of players is static (it is not updated frequently).

- The server does not verify anything about the player before sending back an address to the client. (for example, do not verify that the player is indeed playing, or do not verify access rights or payment for the client).
- The work of the server is then "very short": lookup the movie name in a hashtable and send back the address.

In this case, a single threaded sequential server is appropriate. (loop: accept connection, lookup address, send back reply)

Note: if there can be very many client requests (millions) - the best scalability solution would be to run several instances of the same single threaded server on different machines and to add a router solution in front of them (load balancing).

<u>Assumptions 2</u> If any of the following condition holds:
- There can be a large number of players.
- The set of players is updated frequently (new players are added or removed frequently).
- The server verifies that players are operating correctly in the background.
- The server verifies access rights for the client in a database or payment conditions.

Then the handling of the client request can be long and heavy. In this case a reactor design is more appropriate (thread pool, selector, acceptor, work queue).

Note: it is never correct in this scenario to use a thread-per-connection server because:
i. We do not control how many clients will connect
ii. In any case, the work required for a single connection is "short" (even under assumption 2) compared to the cost of creating a thread. (In the application protocol, we only reply to a single message - there is no conversation).

סעיף ב (5 נקודות)

Key points from the description of the protocol:
- one player plays to more than one receiver
- one player sends the same content to all receivers
- since the player plays regardless of whether receivers are connected, we are working in "streaming" mode - that is, the receiver plays the chunks of data as it receives it. This is in contrast to "file exchange" mode - where the receiver would download a file then play it once it has arrived.

Key points about video playing:
- video files are large (many megabytes) - even for highly compressed formats.
- video is made up of several units (chunks) which contain one or several frames (pictures) and the corresponding audio data.
- in streaming mode, we want to be able to play data without keeping a large buffer on the receiver: this means we must start showing video frames as soon as one or a few video units have arrived.

The most appropriate protocol in these settings is: UDP in multicasting:

- Multicasting allows the player to send video data only once and have all receivers receive it with little overhead on the network.
- UDP is appropriate because:
    - There is no multicast TCP option
    - There is no need for connection handling
    - There is no need for acknowledgments being sent from the receiver to the player (one way communication)
    - Packets should be sent by the player before any ack is received (pipeline)
    - The receiver can deal with some loss of data and loss of order of the packets (assuming packets are sent with redundant information
    - checksum plus possibility to resend part of the data in several packets - and and with packet numbering).

סעיף ג (7 נקודות)

With the new request, streaming is not possible. If taken literally this means the player must now have:

- Server model: a thread per connection model would be appropriate - one thread per receiver connected.
  Note that a reactor would not improve much the situation.
  Note also that naturally we must impose a limitation on the number of clients accepted - it is better to refuse new connections than to crash.
- Connection model: UDP unicast. Not that TCP is still not appropriate for most of the reasons listed above.

Would you accept to implement this modification:
The rational answer is to propose a price formula (not to say "no").
The price of the request is that each connection now takes the same resources on disk access, CPU and network bandwidth as a single movie in the previous model for the duration of the movie.
So the price of a single movie in the new model must cover the price we would obtain in the previous model. If we can charge this or more - then it is worth it.
Alternative solution: propose a way to queue requests for a movie for a while (maximum delay acceptable to a customer), then multicast to all the clients who have sent their requests within this time window. If there are many requests for the same movie that arrive around the same time period - we would save a lot.

שאלה 5 (13 נקודות)

סעיף א (6 נקודות)

1.א

```
A
B
A B
B A
כלום
```

2.א

```
A
B
A B
B A
כלום
```

סעיף ב (7 נקודות)

```
public static void main(String[] args) throws Exception {
        DatagramSocket socket = new DatagramSocket(4445);
        DatagramSocket mcSocket = new DatagramSocket();
        InetAddress mcAddress = InetAddress.getByName("124.1.1.1");
        while (true)  {
            byte[] buf = new byte[256];
            DatagramPacket packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            String message = new String(packet.getData());
            System.out.println(message);

            // send the packet to the multicast group
            mcSocket.send(new DatagramPacket(packet.getData(),packet.getData().length,
                                                mcAddress,3000));
        }
  }
```

שאלה 6 (10 נקודות)

סעיף א (5 נקודות)

```
SELECT Cows.NickName
FROM   Cows,Production
WHERE  Cows.ID = Production.ID AND
       Production.DayInYear = 34 AND
       Production.Year = 2003 AND
       MilkQuantity > 100
```

סעיף ב (5 נקודות)

אפשרות א

Cows

ID
NickName
BornDate

Primary Key: ID
Unique Index: NickName

כאשר:
ID – מספר הזיהוי של הפרה
NickName – שם החיבה של הפרה
BornDate – תאריך הולדתה

Production

ID
DayInYear
Year
MilkQuantity

Primary Key: ID + DayInYear + Year
Foreign Key: ID

כאשר:
ID – מספר הזיהוי של הפרה
DayInYear – יום בשנה [1-365]
Year – שנה
MilkQuantity - תפוקת הפרה בליטרים ליום זה

Weight

ID
DayInYear
Year
Weight

Primary Key: ID + DayInYear + Year
Foreign Key: ID

כאשר:
ID – מספר הזיהוי של העגל
DayInYear – יום בשנה [1-365]
Year – שנה
Weight - משקל העגל

אפשרות ב

Cows

ID
NickName
BornDate
Type

Primary Key: ID
Unique Index: NickName

Production

כאשר:
ID – מספר הזיהוי של הפרה

ID
DayInYear
Year
Data

Primary Key: ID + DayInYear + Year
Foreign Key: ID

כאשר:
ID – מספר הזיהוי של הפרה
DayInYear – יום בשנה [1-365]
Year – שנה
Data - תפוקת חלב (עבור פרות) או משקל (עבור עגלים) ליום זה