גיליון תשובות

מספר נבחן:_____

Q1	Q2
Q3	Q4
Q5	Q6
Q7	TOTAL

שאלה 1 (22 נקודות)

סעיף א (7 נקודות)

נניח כי קיימים שני מופעים v1 ו v2 של המחלקה MyVector החשופים בפני שני אובייקטים אקטיביים T1 ו T2.
אם T1 מבצע:

```
v1.getEqualObjectsNum(v2);
```

ו T2 מבצע:

```
V2.getEqualObjectsNum(v1);
```

ייתכן סדר הפעולות הבא:
T1 'נכנס' לשיטה המסונכרנת **getEqualObjectsNum** של v1 תוך נעילתו.
T2 'נכנס' לשיטה המסונכרנת **getEqualObjectsNum** של v2 תוך נעילתו.
T1 מנסה לבצע את שורת הקוד **synchronized(vector)** (כאשר הפרמטר vector הוא v2) אך נחסם עקב נעילת v2 על ידי T2.
T2 מנסה לבצע את שורת הקוד **synchronized(vector)** (כאשר הפרמטר vector הוא v1) אך נחסם עקב נעילת v1 על ידי T1.

סעיף ב (5 נקודות)

השיטה **GetEqualObjectsNum** אינו בטוחה, כיוון שהיא מאפשרת התערבות של ת'רדים אחרים בפרמטר vector תוך כדי ביצועה – בין בדיקת הגודל של vector ללולאת ה for, ובין בדיקת הערך של כל איבר ואיבר ב vector.
לדוגמא:
נניח כי קיימים שני מופעים v1 ו v2 של המחלקה MyVector החשופים בפני שני אובייקטים אקטיביים T1 ו T2. ב v1 קיים אבר אחד עם ערך 3 וב v2 קיים אבר אחד עם ערך 2.
אם T1 מבצע:

```
v1.getEqualObjectsNum(v2);
```

ו T2 מבצע:

```
v2.remove(0);
```

ייתכן סדר הפעולות הבא:
T1 'נכנס' לשיטה המסונכרנת **getEqualObjectsNum** של v1 תוך נעילתו.
T1 בודק את תנאי ההתחלה – מספר האברים ב v1 זהה למספר האברים ב v2 תוך שחרורו הנעילה של v2.
T2 מבצע את **v2.remove(0)**
T1 מניח כי תנאי ההתחלה מתקיימים ומנסה להשוות את ערך האבר הראשון בכל ווקטור.

סעיף ג (10 נקודות)

<u>אפשרות א</u>: שימוש ב versioned iterator

```
import java.util.Vector;

class MyVector
{

      protected Vector data;
      protected int version;

      public synchronized int  getVersion() {
         return version;
      }

      public synchronized int  size() {
         return data.size();
      }

      public synchronized void add(Object obj) {
         version++;
         data.add(obj);
      }

      public synchronized void remove(int index) {
         version++;
         data.remove(index);
      }

      public synchronized Object get(int index) {
         return data.get(index);
      }

      public synchronized int getEqualObjectsNum(MyVector vector)
         throws Exception
      {
         int currect_version;
         synchronized(vector) {
            currect_version = vector.getVesrion();
            if (size() != vector.size())
               throw new Exception();
         }
         int EqualObjectsNum = 0;
         for (int i = 0; i < size(); i++) {
            synchronized(vector) {
               if (currect_version != vector.getVersion())
                  throw new Exception();
               if (get(i).equals(vector.get(i)))
                  EqualObjectsNum++;
            }
```

```
        }
        return EqualObjectsNum;
    }
}
```

אפשרות ב: שימוש ב snapshot כלומר העתקה מסונכרנת של הווקטור, וביצוע השיטה על העותק של הווקטור.

שאלה 2 (10 נקודות)

סעיף א (3 נקודות)

תכונה נשמרת: במצב הפנימי של המחלקה מספר שלם זוגי - כלומר `getJ()` מחזירה תמיד מספר זוגי.
תרחיש בו התכונה לא נשמרת:
נניח כי קיים מופע e של המחלקה even, בעל מצב פנימי תקין 2, החשוף בפני שני אובייקטים אקטיביים T1 ו T2.
אם T1 מבצע:

```
e.next();
```

ו T2 מבצע:

```
e.next();
```

ייתכן סדר הפעולות הבא:
T1 מתחיל לבצע את next():

```
j_++;
```

T2 מבצע את כל next() כלומר:

```
j_++;
j_++;
```

T1 ממשיך לבצע את next():

```
j_++;
```

התוצאה: במצב הפנימי של e יש מספר לא זוגי.

סעיף ב (7 נקודות)

המחלקה safe אינה בטוחה כי הוא חושפת את השדה הפנימי שלה – even - שאינו בטוח.
לדוגמא:
נניח כי קיים מופע s של המחלקה safe, עם שדה פנימי even בעל מצב פנימי תקין 2, החשוף בפני שני אובייקטים אקטיביים T1 ו T2.
אם T1 מבצע:

```
s.getE().next();
```

ו T2 מבצע:

```
s.getE().next();
```

ייתכן סדר הפעולות הבא:
T1 מקבל reference ל even ומתחיל לבצע עליו את next():

```
j_++;
```

T2 מקבל reference ל even ומבצע עליו את כל next() כלומר:

```
j_++;
j_++;
```

T1 ממשיך לבצע את next():

```
j_++;
```

התוצאה: התכונה לא נשמרת עבור המצב הפנימי של safe - ב even יש מספר לא זוגי.

שאלה 3 (10 נקודות)

סעיף א (5 נקודות)

הדפסה לא חוקית של מקום משוחרר במחסנית.

[ספציפית לקוד הנתון הפונקצייה ()main תדפיס:

8 8

הבעיה נובעת מכך ש ()makeA מחזירה מצביע למקום ב activation frame שלה במחסנית. על פי ה stack model ועבור הביצוע הספציפי של ()main:

```
A* pa = makeA(7);
A* pb = makeA(8);
```

pa יצביע לתוכן של pb ו ()getI שלו יחזיר 8]

סעיף ב (5 נקודות)

()makeA מייצרת זיכרון על ה heap ומחזירה מצביע אליו:

```
#include <iostream>

class A {
public:
   A(int i) { i_ = i; }
   int getI() {return i_;}
private:
   int i_;
};

A* makeA(int j) {
  return new A(j);
}

void main()
{
   A* pa = makeA(7);
   A* pb = makeA(8);
   std::cout << pa->getI() << " " << pb->getI() << std::endl;
   delete pa;
   delete pb;
}
```

שאלה 4 (15 נקודות)

סעיף א (7 נקודות)

```
A 20
A 30
A 200
A 200
```

סעיף ב (8 נקודות)

doA() אינה יעילה:
- הפרמטר v עובר by value
- הערך (מסוג A) אליו מצביע האיטרטור it מועתק (למשתנה current) ומוחזר לחינם.

תיקון:

```
void doA(std::vector<A>& v) {
  std::vector<A>::iterator i;
  for (i = v.begin() ; i != v.end(); ++i) {
    std::cout << "A " << (*i).get(0) << std::endl;
    (*i).set(0, 200);
  }
  for (i = v.begin() ; i != v.end(); ++i) {
    std::cout << "A " << (*i).get(0) << std:endl;
  }
}
```

שאלה 5 (22 נקודות)

סעיף א (3 נקודות) 1

סעיף ב (3 נקודות) TCP

סעיף ג (6 נקודות)

Unicast:	Yes
Duplex communication:	Yes
Infallible:	No
Reliability:	Yes
Checksum:	Yes
Retransmissions:	Yes

סעיף ד (10 נקודות)

```
import java.io.ObjectOutputStream;
import java.io.ObjectInputStream;
import java.net.Socket;

public class client {
  public static void main(String[] args) throws Exception {
    Socket s = null;

    IncNumber element = new IncNumber(2,7);

    Socket server = new Socket(args[0], 2000);
    ObjectOutputStream out =
       new ObjectOutputStream(server.getOutputStream());

    out.writeObject(element);

    ObjectInputStream in =
       new ObjectOutputStream(server.getInputStream());
```

```
    element = (IncNumber)in.ReadObject();

    System.out.println(element.getNum());

    server.close();
    in.close();
    out.close();
  }
}
```

```
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.ServerSocket;

public class server {
  public static void main(String[] args) throws Exception {

    ServerSocket serverSocket = new ServerSocket(2000);
    while (true) {
      Socket client = serverSocket.accept();
      ObjectInputStream in =
         new ObjectInputStream(client.getInputStream());
      IncNumber element = (IncNumber) in.readObject();

      for (int i=0; i<3; i++)
         element.increase();

      ObjectOutputStream out =
         new ObjectOutputStream(client.getOutputStream());
      out.WriteObject(element);

      out.close();
      in.close();
      client.close();
    }
  }
}
```

שאלה 6 (9 נקודות)

Quality of Service(yes/no)	Persistence (yes/no)	Push/Pull	
Yes	Yes	Push	אפליקציה 1
No	Yes	Pull	אפליקציה 2
Yes/No	Yes/No	Push	אפליקציה 3

שאלה 7 (12 נקודות)

סעיף א (6 נקודות)

עבור הספריה UserInterface מתאים Implicit dynamic linking :

- כיוון שהשימוש בו הוא תמידי אין בעיה שיתפוס מקום בזיכרון.

- עדכון של ה UserInterface הוא למעשה שדרוג של התוכנה וניתן לדרוש את הפעלתה מחדש.
- מימוש של ה UserInterface עבור explicit dynamic linking הוא מסורבל, כיוון שממשק הספרייה מכיל פונקציות רבות המופעלות במספר רב של תרחישים.

עבור הספרייה ReportGenerator מתאים Explicit dynamic linking:
- השימוש בספריה הוא נקודתי, פעם בחודש ולכן אין טעם להשאיר אותה בזיכרון – היא תיטען on demand ותוסר מהזיכרון בגמר הפעולה.
- ניתן להוסיף פורמטים שונים של דוחות מבלי לסגור את התוכנה.
- הממשק של הדוחות הוא פשוט וקל למימוש עבור explicit dynamic linking.

סעיף ב (6 נקודות)

- בזבוז של זיכרון: חלקים שונים של UserInterface ושל ה ReportGenerator ניתנים לשימוש על ידי תהליכים אחרים - בלינק סטטי הם יטענו בניפרד עבור כל תהליך.
- בזבוז של מקום על הדיסק: חלקים שונים של UserInterface ושל ה ReportGenerator ניתנים לשימוש על ידי תהליכים אחרים – בלינק סטטי הם יהיו חלק מקובץ ההרצה של כל תהליך.
- עדכון של הספריות UserInterface ו/או ReportGenerator יצריך קימפול מחדש של MovieScheduler.