אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

רשמו תשובותיכם בשאלון זה בלבד

ובמקום המוקצה לכך בלבד !

תשובות מחוץ לשאלון לא יבדקו.

בהצלחה!

תאריך הבחינה: 13.7.2008
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: ב'
משך הבחינה: שלוש שעות
חומר עזר: אסור

שאלה 1	30 נקודות)

סעיף א' (8 נקודות):

המחלקה Consumer מממשת את ה-design pattern של producer-consumer. בהינתן ה-main הבא, השלימו את הריבועים החסרים בעזרת מנגנון wait ו-notify.

הערה: הוספה לתור מתבצעת ע"י השיטה offer(E).

```
import java.io.IOException;

public class Main {
      public static void main(String [] args)
         throws IOException, InterruptedException {
            Consumer<Integer> c = new Consumer<Integer>();
            Thread t = new Thread(c);
            t.start();
            for (int i=0; i<10; i++){
                  c.addElement(new Integer(i));
                  Thread.sleep(1000);
            }
      }
}

import java.util.Queue;
import java.util.concurrent.ConcurrentLinkedQueue;

public class Consumer<E> implements Runnable {
      Queue<E> _queue;
      public Consumer() {
            _queue = new ConcurrentLinkedQueue<E>();
      }
      public void addElement(E e) {
```

```
          synchronized (_queue) {
                _queue.offer(e);
                _queue.notifyAll();
          }
```

```
      }
      private void consume() {
            synchronized (_queue) {
                  while (_queue.isEmpty()) {
                        try {
                              _queue.wait();
                        } catch (Exception ignored) { }
                  }
                  E e = _queue.remove();
                  System.out.println("Consumed " + e);
            }
      }
```

```
	public void run() {
		while (true) {
			consume();
		}
	}
}
```

סעיף ב' (4 נקודות):

Pipe הינו אוביקט בעל שני channels, source ו- sink. אל ה-sink ניתן לכתוב בתים, ומה-source ניתן לקרוא בתים. source ו- sink הינם channels לכל דבר ב-java. במימוש הבא ל producer-consumer החלטנו להשתמש ב-Pipe על מנת לסמן הוספת איברים אל התור. כל פעם שמוסיפים איבר אל התור, יש לסמן זאת על ידי כתיבת בית אחד כלשהו ל-sink. זאת במקום להשתמש ב notify ואין להשתמש ב synchronized בכלל בפתרונכם. עם זאת, על הפתרון להיות thread safe.

השלימו את addElement:

הערות:

(א) ל- source ו- sink שיטות read ו-write, בהתאמה, שמקבלות ByteBuffer כארגומנט.

(ב) השיטות הנ"ל יכולות לזרוק IOException.

(ג) הערך המוחזר הינו מספר הבתים שנקראו/נכתבו.

(ד) מחלקת ה-Main נשארת ללא שינוי מסעיף א'.

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.Pipe;
import java.util.Queue;
import java.util.concurrent.ConcurrentLinkedQueue;

public class Consumer<E> implements Runnable {
	Queue<E> _queue;
	Pipe _pipe;
	public Consumer() throws IOException {
		_queue = new ConcurrentLinkedQueue<E>();
		_pipe = Pipe.open();
		_pipe.sink().configureBlocking(true);
		_pipe.source().configureBlocking(true);
	}

	public void addElement(E e) {
		ByteBuffer buf = ByteBuffer.wrap(new byte[] { 1 });
```

```
                _queue.offer(e);
                  try {
                          _pipe.sink().write(buf);
                  } catch (IOException e1) {}
```

```
      }

      private void consume() throws IOException {
            ByteBuffer buf = ByteBuffer.allocate(1);
            if (_pipe.source().read(buf) == 1 && !_queue.isEmpty()){
                  E e = _queue.remove();
                  System.out.println("Consumed " + e);
            }
      }
      public void run() {
            while (true) {
                  try {
                        consume();
                  } catch (IOException ignored) {}
            }
      }
}
```

<u>סעיף ג' (8 נקודות):</u>

נרצה לשנות את ה-consumer כך שיתמוך במספר תורים שונים. לשיטה addElement נוסיף ארגומנט המציין לאיזה תור מוסיפים את האיבר החדש (ע"י אינדקס התור). כדי לדעת לאיזה תור נוסף האיבר החדש, במקום לרשום 1 אל ה-sink, רשמו את אינדקס התור הרלוונטי. השלימו את השיטה addElement:

```
import java.io.IOException;

public class Main {
      public static void main(String [] args)
          throws IOException, InterruptedException{
            Consumer<Integer> c = new onsumer<Integer>(10);
            Thread t = new Thread(c);
            t.start();
            for (int i=0; i<10; i++){
                  c.addElement(new Integer(i), i);
                  Thread.sleep(1000);
            }
      }
}
```

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.Pipe;
import java.util.Queue;
import java.util.Vector;
import java.util.concurrent.ConcurrentLinkedQueue;

public class Consumer<E> implements Runnable {
	Vector<Queue<E>> _queues;
	Pipe _pipe;
	// at most 256 queues
	public Consumer(int queueNum) throws IOException {
		_queues = new Vector<Queue<E>>(queueNum);
		//create all the queues
		for (int i = 0; i < queueNum; i++)
			_queues.add(new ConcurrentLinkedQueue<E>());

		//create the pipe
		_pipe = Pipe.open();
		_pipe.sink().configureBlocking(true);
		_pipe.source().configureBlocking(true);
	}
	public void addElement(E e, int whichQueue) {
		if (whichQueue >= _queues.size())
			throw new IndexOutOfBoundsException();
```

```
ByteBuffer buf = ByteBuffer.wrap(new byte[]
	{ (byte) whichQueue });
_queues.get(whichQueue).offer(e);
try {
	_pipe.sink().write(buf);
} catch (IOException e1) {}
```

```
	}
	private void consume() throws IOException {
		ByteBuffer buf = ByteBuffer.allocate(1);
		if (_pipe.source().read(buf) == 1) {
			buf.flip();
			int whichQueue = buf.get();
			if (!_queues.get(whichQueue).isEmpty()) {
				E e = _queues.get(whichQueue).remove();
				System.out.println(
				 "Consumed " + e + " from queue "+
				 whichQueue);
			}
		}
	}
```

```
	public void run() {
		while (true) {
			try {
				consume();
			} catch (IOException ignored) {}
		}
	}
}
```

סעיף ד' (10 נקודות):

הפיתרון של סעיף ג' כרוך בכמה בעיות. לדוגמא, מכיוון שכולם כותבים לתוך אותו Pipe, שבסופו של דבר גורר system call, מתבצעת סראליזציה של הוספת האיברים לתורים. בכדי לפתור את הבעייה, נרצה להחזיק Pipe שונה לכל תור. על מנת להאזין לכל ה Pipe-ים במקביל, נעזר ב-Selector . השלימו את השיטות addElement ו-consume.

```
public class Main {
	public static void main(String [] args)
	    throws IOException, InterruptedException {
		Consumer<Integer> c = new Consumer<Integer>(10);
		Thread t = new Thread(c);
		t.start();
		for (int i=0; i<10; i++){
			c.addElement(new Integer(i), i);
			c.addElement(new Integer(i+2), i);
			Thread.sleep(1000);
		}
	}
}
public class Consumer<E> implements Runnable {
	Vector<Queue<E>> _queues;
	Pipe[] _pipes;
	Selector _selector;
	class Attachment {
		public Queue<E> queue;
		public Pipe pipe;
		int index;
		public Attachment(Queue<E> queue, Pipe pipe, int index) {
			this.queue = queue;
			this.pipe = pipe;
			this.index = index;
		}
	}
	public Consumer(int queueNum) throws IOException {
		_queues = new Vector<Queue<E>>(queueNum);
		for (int i = 0; i < queueNum; i++)
			_queues.add(new ConcurrentLinkedQueue<E>());
		_selector = Selector.open();
		// create all pipes, one pipe per queue
		_pipes = new Pipe[queueNum];
		for (int i = 0; i < queueNum; i++) {
			_pipes[i] = Pipe.open();
			Pipe pipe = _pipes[i];
			pipe.sink().configureBlocking(true);
			pipe.source().configureBlocking(false);
			// register the source in the selector, and attach
			// an appropriate object
			pipe.source().register(_selector, SelectionKey.OP_READ,
				new Attachment(_queues.get(i), pipe, i));
		}
	}
```

```
public void addElement(E e, int whichQueue) {
      if (whichQueue >= _queues.size())
            throw new IndexOutOfBoundsException();
      ByteBuffer buf = ByteBuffer.wrap(new byte[] { 1 });
            _queues.get(whichQueue).offer(e);
            try {
                    _pipes[whichQueue].sink().write(buf);
            } catch (IOException e1) {}
}
private void consume() throws IOException {
  ByteBuffer buf = ByteBuffer.allocate(1);
  if (_selector.select() != 0) {
    Iterator<SelectionKey> it =_selector.selectedKeys().iterator();
    while (it.hasNext()) {
      SelectionKey key = it.next();
      it.remove();
            Attachment att = (Attachment) key.attachment();
            Queue<E> queue = att.queue;
            if (!queue.isEmpty()) {
                  E e = queue.remove();
                  System.out.println("Consumed " + e +" from queue " + att.index);
            }
            att.pipe.source().read(buf);
            buf.flip();
      }
  }
}
public void run() {
      while (true) {
            try {
                  consume();
```

```
                } catch (IOException ignored) {}
            }
        }
}
```

שאלה 2	30 נקודות)

סביבת ריצה מבוזרת כוללת בין השאר מערכת קבצים מבוזרת. בשאלה זו נממש מערכת כזו באמצעות RMI. תכנון המערכת כולל שני אלמנטים מרכזיים: RemoteDisk ו-RemoteFile:

```
public interface RemoteDisk extends java.rmi.Remote {
      /**
       * @param filePath the path of the file on the remote disk
       * @return a remote reference to the file object
       * @throws IOException
       * @throws java.rmi.RemoteException
       */
      public RemoteFile open(String filePath) throws IOException,
                  java.rmi.RemoteException;

      /** close the file
       * @param file
       * @throws IOException
       * @throws java.rmi.RemoteException
       */
      public void close(RemoteFile file) throws IOException,
          java.rmi.RemoteException;
}

public interface RemoteFile extends java.rmi.Remote {
      /**
       * @param buf -the byte source
       * @return how many bytes were written
       * @throws IOException
       */
      public int write(byte[] buf) throws IOException,
          java.rmi.RemoteException;

      /**
       * @param buf- where to put the data
       * @return how much bytes were read
       * @throws IOException
       */
      public int read(byte[] buf) throws IOException,
         java.rmi.RemoteException;

      /**
       * @return the file path
       * @throws java.rmi.RemoteException
       */
      public String getPath() throws java.rmi.RemoteException;
}
```

RemoteDisk מספק אבסטרקציה ל-דיסק מרוחק, ו-RemoteFile לקובץ מרוחק.
RemoteDisk מאפשר לבקש קובץ על פי שם ומחזיר אוביקט שמממש RemoteFile.
RemoteFile מבצע כתיבה וקריאה לקובץ פיזי אשר נמצא על מחשב מרוחק. להלן דוגמת

שימוש במערכת. הדוגמא מניחה כי שרת RemoteDisk נרשם ב rmiregistry במחשב המקומי תחת השם "remoteDisk", מבקשת לפתוח קובץ בשם "/tmp/ppp10", וכותבת לתוכו את המחרוזת "hello".

```
public class FileTest {
      public static void main(String[] args) throws
            NotBoundException, IOException {
            RemoteDisk rd = (RemoteDisk) Naming
                        .lookup("//localhost:9091/remoteDisk");
            RemoteFile rf = rd.open("/tmp/ppp10");
            rf.write("hello".getBytes());
            rd.close(rf);
      }
}
```

סעיף א' (4 נקודות):

השלימו את הקוד הבא, הממש RemoteFile ו-RemoteDisk

הערות:

(א) כתיבה ל-File מתבצעת על ידי FileOutputStream, שמקבל את ה-File בבנאי ותומך בשיטה write(byte []).

(ב) קריאה מ-File מתבצעת בצורה דומה על ידי FileInputStream.

```
public class RemoteFileImpl extends java.rmi.server.UnicastRemoteObject
            implements RemoteFile {
       // The local file object, through which we do all IO
      File _file;
      String _filePath;
      protected RemoteFileImpl(String filePath) throws IOException {
            super();
            _file = new File(filePath);
            _file.createNewFile();
            _filePath = filePath;
      }
      public int read(byte[] buf) throws IOException, RemoteException {
            FileInputStream fis = new FileInputStream(_file);
            return fis.read(buf);
      }
      public int write(byte[] buf) throws IOException, RemoteException {
            FileOutputStream fos = new FileOutputStream(_file);
            fos.write(buf);
            return buf.length;
      }
      public String getPath() throws RemoteException {
            return _filePath;
      }
}
```

```
public class RemoteDiskImpl extends java.rmi.server.UnicastRemoteObject
            implements RemoteDisk {
       // cache opened files in a hash map.
      HashMap<String, RemoteFile> _files = new HashMap<String, RemoteFile>();
      protected RemoteDiskImpl() throws RemoteException {
            super();
      }
      public RemoteFile open(String filePath)
          throws IOException, RemoteException {
            if (_files.containsKey(filePath))
                  return _files.get(filePath);
            else {
                  RemoteFileImpl rf = new RemoteFileImpl(filePath);
                  _files.put(filePath, rf);
                  return rf;
            }
      }
      public void close(RemoteFile file) throws IOException, RemoteException{
            _files.remove(file.getPath());
      }
}
```

<u>סעיף ב' (10 נקודות):</u>

בהינתן מחלקת ה-`FileTest` לעיל, כמה פעולות תקשורת מתבצעות במהלך השיטה `main`? ציינו את הסה"כ ולאחר מכן פרטו.

1. התחברות לname-server (1 הלוך-חזור) – פעולת lookup
2. קריאה של open על אוביקט מרוחק rd: `rd.open("/tmp/ppp10")` (הפעולה הזו מחזירה `remote reference` לאוביקט מרוחק `rf`)
3. קריאה של write על אוביקט מרוחק `rf`: `rf.write("hello".getBytes())`
4. קריאה של close על אוביקט מרוחק `rd`

סה"כ: 4 הלוך-חזור

הערה: אפילו קריאה ל-method מסוג void גורמת להלוך-חזור בתקשורת. הסיבה היא שה-client מצפה לאישור שהקריאה הסתיימה בהצלחה (באופן סינכרוני) או ל-exception במקרה של תקלה.

<u>סעיף ג' (6 נקודות):</u>

לאחר שנמצא כי המערכת אינה עומדת בעומסים, הוחלט להרחיב את הממשק ולתמוך ב `load balancing` בין הדיסקים. הדיסקים נרשמים אצל ה `RemoteDiskBalancer` והיא בוחרת איזה דיסק ישמש לפתיחת הקובץ באופן אקראי. לצורך כך, הוגדר בנוסף הממשק הבא, ולא שונו המימושים של המחלקות הקודמות:

```
public interface RemoteDiskBalancer extends Remote {
      /** locate a remote disk
       * @return a remote reference to the disk
       * @throws java.rmi.RemoteException
       */
      public RemoteDisk findAFreeDisk() throws  java.rmi.RemoteException;
      public void register(RemoteDisk rd) throws  java.rmi.RemoteException;
}
```

```
public class RemoteDiskBalancerImpl extends
java.rmi.server.UnicastRemoteObject implements RemoteDiskBalancer {
      Vector<RemoteDisk> _disks = new Vector<RemoteDisk>();
      protected RemoteDiskBalancerImpl() throws RemoteException {
            super();
      }
      private static final long serialVersionUID = 1L;
      public RemoteDisk findAFreeDisk() throws RemoteException {
            if (_disks.isEmpty())
                  return null;
            return _disks.get((int)Math.random()*_disks.size()));
      }
      public void register(RemoteDisk rd)
        throws RemoteException {
            _disks.add(rd);
      }
}
```

המחלקה הבאה רושמת RemoteDiskBalancer חדש לשרת השמות הרץ על המחשב הלוקלי:

```
public class Main {
      public static void main(String [] args)
          throws RemoteException, MalformedURLException {
            RemoteDiskBalancer rd = new RemoteDiskBalancerImpl();
            Naming.rebind("//localhost:9091/remoteDiskBalancer", rd);
      }
}
```

המחלקה הבאה רושמת דיסק ל-RemoteDiskBalancer :

```
public class DiskRegister {
      public static void main(String[] args)
          throws RemoteException, MalformedURLException, NotBoundException {
            RemoteDisk rd = new RemoteDiskImpl();
            RemoteDiskBalancer rdb = (RemoteDiskBalancer)Naming.lookup(
               "//localhost:9091/remoteDiskBalancer");
            rdb.register(rd);
      }
}
```

בהנחה שהאוביקט היחידי הרשום ל-rmiregistry הינו "remoteDiskBalancer", ושתי המחלקות לעיל הורצו בהצלחה, השלימו את הקוד הבא, אשר פותח קובץ בשם "/tmp/ppp12" ורושם לתוכו "goodbye":

```
public class FileTest {
      public static void main(String[] args)
          throws NotBoundException, IOException {
            RemoteFile rf;
            RemoteDisk rd;
            RemoteDiskBalancer rdb = (RemoteDiskBalancer)Naming.lookup(
                 "//localhost:9091/remoteDiskBalancer");
           rd = rdb.findAFreeDisk();
           rf = rd.open("/tmp/ppp10");
           rf.write("goodbye".getBytes());
```

```
            rf.write("hello".getBytes());
            rd.close(rf);
        }
}
```

<u>סעיף ד' (10 נקודות):</u>

בהינתן מחלקת ה-FileTest לעיל, כמה פעולות תקשורת מתבצעות במהלך השיטה main? ציינו את הסה"כ ולאחר מכן פרטו.

1. פניה ל-name server לצורך בנייה של ה-rdb disk balancer.
2. פניה ל-rdb לצורך איתור של דיסק פנוי (rdb.findAFreeDisk();) הערך המוחזר הוא remote reference לאוביקט מסוג remote disk.
3. פניה ל-rd לצורך פתיחה של קובץ (rd.open();) הערך המוחזר הוא remote reference לאוביקט מסוג remote file.
4. קריאה לwrite (1 או 2 פעמים)
5. קריאה ל-close

סה"כ 5 הלוך-חזור

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. בנוסף יש לציין אם התוכנית כולה (ללא קשר למיקום תמונת הזכרון הרצוי) מבצעת פעולת זכרון לא רצויה<u>ואיזה</u> או לא.

<u>להלן דוגמא</u>

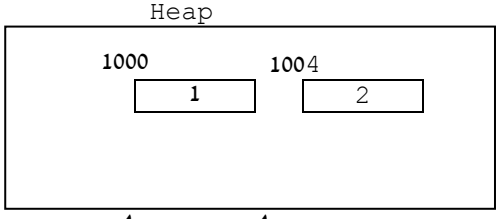
```
{
    int i=3;
    int *p1 = new int(1);
    int *p2 = new int(2);
}
```

תמונת הזכרון כאן: ←

Stack

884	
888	
892	1004
896	1000
900	3

Heap



האם התוכנית מבצעת פעולת זיכרון לא רצויה: <u>כן – דליפת זכרון.</u>

דגשים

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

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

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

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

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

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

סעיף א

```
void main() {
      SmartPointer s1(new int(3));
      SmartPointer s2(s1);
      SmartPointer s3(s2);
```

סעיף א - תמונת הזכרון כאן: ←

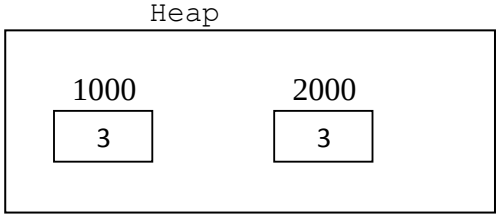
```
}
```

פתרון סעיף א, תמונת הזכרון:

Stack

880	2000
884	1000
888	2000
892	1000
896	2000
900	1000

Heap



האם התוכנית מבצעת פעולת זיכרון לא רצויה: **לא**

סעיף ב

```
void main()
{
      int *pi = new int(3);
      SmartPointer s1(pi);
      SmartPointer s2(pi);
```

סעיף ב - תמונת הזכרון כאן: ←

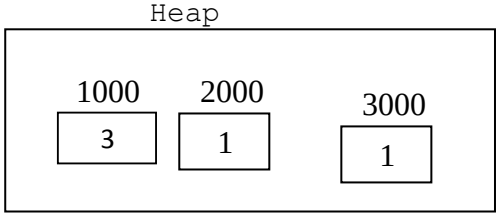
```
      delete pi;
}
```

פתרון סעיף ב, תמונת הזכרון:

Stack

880	
884	3000
888	1000
892	2000
896	1000
900	1000

Heap



האם התוכנית מבצעת פעולת זיכרון לא רצויה: כן – מחיקת כתובת 1000 3 פעמים

סעיף ג

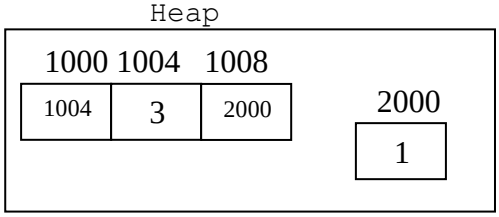
```
class B {
private:
      int *pi;
      int i;
      int *p;
public:
      B();
      ~B();
};
B::B(): i(3), pi(&i), p(new int(1)){}
B::~B() {
      delete pi;
      delete p;
}
void main() {
      B *pb = new B();
                        ←———— סעיף ג- תמונת הזכרון כאן:
      delete pb;
}
```

פתרון סעיף ג, תמונת הזכרון:

Stack

880	
884	
888	
892	
896	
900	1000

Heap



האם התוכנית מבצעת פעולת זיכרון לא רצויה: כן – מחיקת 1004 פעמיים

סעיף ד

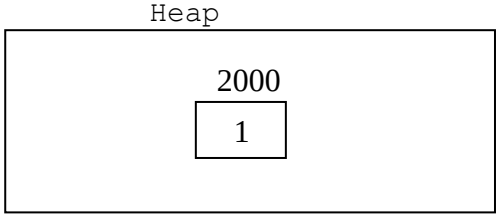
```
class D : public B {
public:
      int b;
};
void main() {
      D d;
      d.b = 5;
            ←———— סעיף ד- תמונת הזכרון כאן:
}
```

פתרון סעיף ד, תמונת הזיכרון:

Stack

880	

Heap



884	
888	5
892	2000
896	3
900	896

האם התוכנית מבצעת פעולת זיכרון לא רצויה: כן – delete למשתנה על המחסנית

שאלה 4	**(14 נקודות)**

בשאלה זו עליכם לממש מחסנית המחזיקה מצביעים ל- int ב C++ תוך שימוש ב stack של STL. הנכם נדרשים לתמוך ב:

(א) ניהול משאב זכרון כלומר בעת הכנסת ערך int המחסנית מקצה זכרון חדש עבורו ומנהלת אותו, משמע משחררת את הזכרון כשצריך.

(ב) תמיכה בפעולת undo. פעולה זו מבטלת הכנסת ערכים או הוצאתם. למשל הכנסת שני ערכים: 1 ואחר"כ 2 ואז בצוע undo יבטל את הכנסת הערך 2. undo נוסף יבטל את הכנסת הערך 1. הכנסת שני ערכים 1 ואחר"כ 2 ואז בצוע pop ואז undo יבטל את פעולת ה pop ובמחסנית יהיו הערכים 1 ו 2 כאילו לא קרה pop.

דגשים

1) יש להשתמש במחסנית של STL ואין להשתמש במבני נתונים או containers אחרים. בפרט אין לממש מבנה נתונים בעצמכם.

2) עליכם למנוע אפשרות העתקה או השמה של המחסנית לאחרת ע"י הצהרה על copy constructor ו- assignment operator כ private.

3) ניתן להניח כי השימוש במחסנית תקין. למשל לא יבוצע pop, top למחסנית ריקה. כמו כן לא יבוצע undo כאשר אין יותר מה לשחזר. המנעו מקוד מיותר.

4) הקפידו על פתרון קריא. יש לכתוב קוד רק במקום המוקצה לכך. העזרו במחברת הטיוטה והעתיקו את הפתרון לשאלון רק לאחר שהנכם בטוחים בו.

להלן דוגמת הרצה:

```
void main()
{
  UndoableResourceStack s;
  int *i;
  i = new int(1);
  s.push(i);
  delete i;
  cout << *(s.top()) << endl;
  i = new int(2);
  s.push(i);
  delete i;
  cout << *(s.top()) << endl;
  s.pop();
  cout << *(s.top()) << endl;
  s.undo();
  cout << *(s.top()) << endl;
  s.undo();
  cout << *(s.top()) << endl;
}
```

תוצאת ההרצה:

```
1
2
```

```
1
2
1

#include <stack>
using namespace std;
class UndoableResourceStack
{
private:
      stack<int*> s;
      stack<int*> undoStack;
      UndoableResourceStack (const UndoableResourceStack &o);
      UndoableResourceStack operator=(const UndoableResourceStack &o);

public:
      UndoableResourceStack();
      int* top()const;
      int size() const;
      void push(int *i);
      void pop();
      void undo();
      virtual ~UndoableResourceStack();

};
```

מימוש

```
UndoableResourceStack::UndoableResourceStack() { };
int* UndoableResourceStack::top() const {
      return s.top();
}
int UndoableResourceStack::size() const {
            return s.size();

}
```

```
void UndoableResourceStack::push(int *i)
{
	s.push(new int(*i));

	undoStack.push(0);  // 0 means push

}
void UndoableResourceStack::pop()
{
	undoStack.push(s.top());
	s.pop();

}
void UndoableResourceStack::undo()
{
	int *i = undoStack.top();
	undoStack.pop();
	if (i == 0)
	{
		delete s.top();
		s.pop();
	} else
	{
		s.push(i);
	}

}
UndoableResourceStack::~UndoableResourceStack()
{
	while (s.size() != 0 )
	{
		delete s.top();
		s.pop();
	}
	while (undoStack.size() != 0 )
	{
		delete undoStack.top();
		undoStack.pop();
	}
}
```

שאלה 5	(10 נקודות)

ניתנות ההגדרות הבאות ב-Java:

```
class Resource {
    public String resourceName;
    public int cost;
}
class Consumer {
    public Resource source;
    public String consumerName;
    public Consumer(Resource source, String name) {
        this.source = source; this.consumerName = name;
    }
}
class Producer {
    public Resource target;
    public DateTime creationTime;
    public Producer(String name) {
        this.target = new Resource(name);
        this.creationTime = DateTime.now();
    }
}
class Transaction {
    public Vector<Consumer> consumers;
    public Producer producer;
    public Resource resource;
    public DateTime creationTime;
    public Transaction(String name, Vector<Consumer> consumers) {
        this.consumers = consumers;
        this.producer = new Producer(name);
        this.resource = producer.target;
        this.creationTime = producer.creationTime;
    }
    public lockResource(String name, Resource resource,
                        Vector<Consumer> consumers) {
        this.consumers = consumers;
        this.producer = null;
        this.resource = resource;
        this.creationTime = DateTime.now();
    }
}
```

<u>סעיף א' (4 נקודות):</u>

הגדר אוסף טבלאות המיצג את 4 המחלקות הנ"ל ב-SQL והקשרים בינהן. הגדר את המפתחות הדרושים. על ה-schema להכיל לפחות טבלה אחת לכל מחלקה שתכיל שורה לכל instance של המחלקה.

create table Resource(id int, name varchar(200), cost int)
Primary Key: ID

create table Consumer(id int, source int, name varchar(200))
Primary Key: ID
Foreign Key: source references Resource.id

Create table Producer(id int, target int, creationTime datetime)
Primary Key: ID
Foreign Key: target references Resource.id

create table Transaction(id int, producer int, resource int, creationTime datetime)
Primary Key: ID
Foreign Key: producer references Producer.id
Foreign Key: resource references Resource.id

create table TransactionConsumers(transactionId int, consumerId int, rank int)
Primary Key: (transactionId, resourceId)
Foreign Key: transactionId references Transaction.id
Foreign Key: consumerId references Consumer.id

NOTE: the key elements of the translation Java->SQL are:

- Object definitions become tables
- Primitive values become attributes of the tables
- Object references become foreign keys
- Multiple value fields (such as the vector<Consumer>) become n-n relations that require a cross-table (TransactionConsumers).

Since there was no specification as to the unicity of names, we decided to introduce ID fields as primary keys for all objects.

<u>סעיף ב' (3 נקודות):</u>

כתב את ה-SQL הנדרש להוספת אוביקט מטיפוס Transaction ב-database.

The creation of an object is encoded in Java in the constructor. We therefore must translate the code of the Transaction constructor into SQL code:

The order must be such that we first:

- Create the Producer row and initialize it
- Create the Transaction row
- Link the Transaction to the provided Consumers

The expected parameters are: the name of the transaction (a string) and a vector of consumer object references. Object references in Java are converted to the Primary Key of the objects in SQL. So we expect to get a vector of consumer IDs.
We will call these parameters NAME and CONS[i] (I in 1…n).

```
;; Producer constructor is first invoked
   ;; The Producer constructor invokes the Resource constructor
   ;; Get a new ID not yet used by anyone
   select newResouceId = max(ID)+1 from Resource
   insert into Resource(id, name) values ( newResourceId, NAME )

   ;; Create the Producer
   select newProducerId = max(ID)+1 from Producer
   select now = now()
   insert into Producer(id, target, creationTime)
   values (newProducerId, newResourceId, now)

;; Create the Transaction
select newTransactionId = max(ID)+1 from Transaction
insert into Transaction(id, name, producer, resource, creationTime)
values (newTransactionId, newProducerId, newResourceId, now)

;; Link the Transaction to the consumers
for i = 1 to N
   insert into TransactionConsumers(transactionId, consumerId, rank)
   values (newTransactionId, CONS[i],  i)
```

סעיף ג' (3 נקודות):

כתב שאילתת SQL המחזירה רשימת ה-consumers המחוברים ל-producer מסוים.

The key point is that producers are linked to consumers through transactions in our model.
The select, therefore, requires a join on transactions.

We will call the producer we look for ProducerId.
We are interested in references to consumers – so we want to get a result set of consumerIds.

```
Select TransactionConsumers.consumerId from
  TransactionConsumers left join Transaction
  on TransactionConsumers.transactionId = Transaction.id
where Transaction.producer = ProducerId
```