גיליון תשובות

מספר נבחן:_____

שאלה 1 (30 נקודות)

סעיף א (5 נקודות)

הרצת התוכנית נכונה, שהרי לכ ביצוע מקבילי קיים סדר פעולות סדרתי המגיע לאותה תוצאה.
כעיקרון, חבק אינו בעיה של נכונות אלא של חיות (liveness), אך גם אם נחשיב אותו כבעיית נכונות, ניתן במקרה שלנו למצוא סדר פעולות בו מתקבל החבק גם בחישוב סידרתי:

- ה Producer ממלא את מיכל המים המטוהרים
- ה Consumer מרוקן את מיכל המים המטוהרים
- ה Consumer ממלא את מיכל המים המשומשים
- ה Purifier מרוקן את מיכל המים המשומשים
- ה Producer ממלא את מיכל המים המטוהרים
- ה Consumer מרוקן את מיכל המים המטוהרים
- ה Producer ממלא את מיכל המים המטוהרים
- ה Consumer ממלא את מיכל המים המשומשים
- ה Consumer מרוקן את מיכל המים המטוהרים
- ה Producer ממלא את מיכל המים המטוהרים

כך שאין בעיה של נכונות

סעיף ב (10 נקודות)

מתודת ה run(), של כל אחד משלושת ה Tasks עליהם רצים שלושת הת'רדים, מוגדרת עם לולאת while (true)

- נשנה את תנאי עצירת הלולאה ל while(!Thread.interrupted())
- תפישת InterruptedException מאפסת את השדה, ולכן יש לבצע interrupt() על הת'רד הנוכחי, כדי לשמר את ערך השדה ב Task שמעליו.

פתרון שאינו משתמש במתודות ה interrupt של המחלקה Thread התקבל רק אם הת'רד אכן מפסיק לעבוד כתוצאה מביצוע interrupt(), בין אם הוא ב wait ובין אם לאו.

```
class ProducingTask implements Runnable {
...
  public void run() {
    while (!Thread.interrupted())
      _purifiedPool.add(_purifiedPool.getCapacity());
  }
}

class PurifyingTask implements Runnable {
...
  public void run() {
```

```
    while (!Thread.interrupted())
      _purifiedPool.add(_purifier.purify(_usedPool.remove()));
  }
}
class ConsumingTask implements Runnable {
...
  public void run() {
    while (!Thread.interrupted())) {
      float w = _purifiedPool.remove();
      try { Thread.sleep((long)w); } catch (InterruptedException e) { Thread.currentTrhrea().interrupt(); }
      _usedPool.add(w);
    }
  }
}

class WaterPool implements Pool {
 ...
 public synchronized float remove() {
    while (_content == 0)
      try { wait(); } catch (InterruptedException e) { Thread.currentThread().interrupt(); return 0; }
    float ret = _content;
    _content = 0;
    notifyAll();
    return ret;
  }

  public synchronized void add(float addition) {
    while (_content + addition  > _capacity)
      try { wait(); } catch (InterruptedException e) { Thread.currentThread().interrupt();  return; }
     _content += addition;
    notifyAll();
  }
}
```

סעיף ג (3 נקודות)

קיים מוניטור אחד לשתי ההמתנות (this) כך שביצוע notifyAll() יחזיר למעגל התזמון את כל הת'רדים הממתינים על מוניטור זה.

סעיף ד (12 נקודות)

נגדיר שני מוניטורים שונים, ונעצב את ההמתנה וההערה בהתאם

```
class WaterPool implements Pool {
  private final long _capacity;
  private float _content;
  private Object _monitorInc, _monitorDec;

  WaterPool(float content, long capacity)  {
    _content = content; _capacity = capacity;
    _monitorInc = new Object(); _monitorDec = new Object();
  }

  public long getCapacity() { return _capacity; }
  public synchronized float getContent () { return _content; }
  public synchronized float remove() {
    while (_content == 0)
      synchronized (_monitorInc) {
        try { _monitorInc.wait(); } catch (InterruptedException e) { return 0; }
      }
    float ret = _content;
    _content = 0;
    synchronized (_monitorDec) {
      try {_ monitorDec.notifyAll(); } catch (InterruptedException e) { return; }
    }
    return ret;
  }

  public synchronized void add(float addition) {
    while (_content + addition  > _capacity)
      synchronized (_monitorDec) {
        try { _monitorDec.wait(); } catch (InterruptedException e) { return; }
      }
     _content += addition;
    synchronized (_monitorInc) {
      try {_monitorInc notifyAll(); } catch (InterruptedException e) { return; }
    }
  }
}
```

הערה: בפתרון זה קיימת סכנת deadlock. במועד ג של שנה זו, נתבקשו הסטודנטים להסביר כיצד מתחולל ה deadlock, וכן הם נדרשו לשנות את קוד המחלקה WaterPool, תוך שימוש ב Semaphore, כך שה deadlock ימנע.

אתם מוזמנים לתרגל גם את זה.

שאלה 2 (30 נקודות)

סעיף א (7 נקודות)

5012	1
5008	*vtable
5004	1200
5000	1200

1200	1204	1208
5008	0	0

סעיף ב (8 נקודות)

5020	20
5016	2
5012	*vtable
5008	1000
5004	1200
5000	1400

1200	1204	1208
1000	0	1400

1000	1004
*vtable	1

1400	1404	1408
5012	1200	0

סעיף ג (7 נקודות)

```
EventsStack::EventsStack(const EventsStack &other):_top(0),_last(0) {
	EventNode *cursor = other._last;
	while (cursor != 0) {
		push(cursor->data);
		cursor = cursor->prev;
	}
}
```

סעיף ד (8 נקודות)

```
void EventsStack::pop() {
	if ( _top == 0 ) return;
	EventNode *oldTop = _top;
	_top = _top->next;
	if (_top !=0 ) {
```

```
                _top->prev = 0;
        }
        delete oldTop;
        if (oldTop == _last) {
                _last = 0;
        }
}
```

שאלה 3 (30 נקודות)

סעיף א (6 נקודות)

What must be change in StompOperatorImp to work with a message receiver:
1. Remove the getMessages message implementation.
2. Add a messageReceiver parameter to the constructor.
3. Pass the messageReceiver parameter to the Listener.
4. Change the Listener so that it passes the received messages to the receiver as soon as received.
5. Remove the _msgs member.

```
public class StompOperatorImpl extends java.rmi.server.UnicastRemoteObject , implements StompOperator {
  protected PrintWriter _writer;
  protected BufferedReader _reader;
  private List<String> _msgs;

  // Define a listener task to receive the messages from the Stomp server.
  private class Listener implements Runnable {
    private MessageTokenizer _tok;
    private MessageReceiver _mr;
    public Listener(MessageReceiver mr) {
      _tok = new MessageTokenizer(_reader, '\0');
      _mr = mr;
    }
    public void run() throws IOException {
      while (_tok.isAlive())
        synchronize (_msgs) { _msgs.add(_tok.nextToken());}
        _mr.message(_tok.nextToken());
    }
  }

  public StompOperatorImpl(InputStream in, OutputStream out, MessageReceiver mr)
    throws java.rmi.RemoteException, IOException {
    _reader = new BufferedReader(new InputStreamReader(in,"UTF-8"));
```

```
    _writer = new PrintWriter(new OutputStreamWriter(out,"UTF-8"));
    _msgs = new ArrayList<String>();
    // Construct listener
    new Thread(new Listener(mr)).start();
  }

  public void subscribe (String group)   throws java.rmi.RemoteException, IOException {
      synchronized (_writer) { _writer.print("SUBSCRIBE\ndestination: " + group + "\n\n" + '\0'); }
  }

  public void unsubscribe (String group)  throws java.rmi.RemoteException, IOException {
      synchronized (_writer) { _writer.print("UNSUBSCRIBE\ndestination: " + group+ "\n\n" + '\0'); }
  }

  public void send (String group, String str)   throws java.rmi.RemoteException, IOException {
      synchronized (_writer) { _writer.print("SEND\ndestination: " + group+ "\n\n" + str + "\n" + '\0'); }
  }

  public List<String> getMessages() throws java.rmi.RemoteException {
    // Snapshot copy of the messages and reset it.
    List<String> res;
    synchronized (_msgs) {
      res = new ArrayList<String>(_msgs);
      _msgs.clear();  // reset the accumulator to receive new messages.
    }
    return res;
  }
}
```

סעיף ב (6 נקודות)

The required changes in StompClient are:
- Create a MessageReceiver instance to handle incoming messages.
- Pass this instance as a parameter to the StompConnector.
- Remove the loop that calls getmessages – since messages are now handled by the receiver.

```
public class StompClient {
  public static void main(String[] args) {
    try {
      MessageReceiver mr = new MessageReceiverImpl();
      StompConnector c = (StompConnector)Naming.lookup("rmi://132.23.5.8:2010/StompConnector");
      StompOperator s = c.connect("user", "password",mr);
      s.subscribe("q1");  s.subscribe("q2"); s.send("q3", "Suzy Surprise");
      Thread.sleep(60000);
      for (String s : c.getMessages())
```

```
        System.out.println(s);
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

סעיף ג (10 נקודות)

We want the client to send back to the StompServer a message each time the client receives a message. NOTE: the decision to send back the message MUST be made in the StompClient – it cannot be only in the StompClientRMI process – it must be triggered by code that lives inside the StompClient process. (This is indicated in the question by the sentence "the message is sent from the memory of the StompClient process").

There were several possible answers. The best one is one where ALL the changes are only in the StompClient side – without any changes to the StompClientRMI.

The way for a StompClient to react to the event of a message arriving is to implement a new version of MessageReceiver. So this is where we start.
The way for a StompClient to send a message to the StompServer is to use the StompOperator send() method.
Let's put these 2 things together in this solution:

```
public class MessageReceiverImp2 extends java.rmi.server.UnicastRemoteObject implements MessageReceiver
{
  private StompOperator _sender = null;
  MessageReceiverImpl() throws java.rmi.RemoteException { }
  void setSender(StompOperator s) {_sender = s;}
  public void message(String str) throws java.rmi.RemoteException {
    if (_sender != null) _sender.send("q1", str);
  }
}
```

```
public class StompClient {
  public static void main(String[] args) {
    try {
      MessageReceiverImpl2 mr = new MessageReceiverImpl2();
      StompConnector c = (StompConnector)Naming.lookup("rmi://132.23.5.8:2010/StompConnector");
      StompOperator s = c.connect("user", "password",mr);
      mr.setSender(s);
      s.subscribe("q1");  s.subscribe("q2"); s.send("q3", "Suzy Surprise");
    } catch (Exception e) { e.printStackTrace(); }
  }
}
```

Another acceptable solution (less nice because it causes more changes to more components) was to change the StompClientRMI code and the MessageReceiver interface so that message(str) returns a string instead of being a void method.

```
public interface MessageReceiver extends java.rmi.Remote {
  // If message returns a non-null value, then send this value back to the sender.
  String message(String str) throws java.rmi.RemoteException;
}
```

Then change the Listener class inside StompOperatorImpl:

```
private class Listener implements Runnable {
    private MessageTokenizer _tok;
    private MessageReceiver _mr;
    public Listener(MessageReceiver mr) {
      _tok = new MessageTokenizer(_reader, '\0');
      _mr = mr;
    }
    public void run() throws IOException {
      while (_tok.isAlive()) {
        String answer = _mr.message(_tok.nextToken());
       If (answer != null)
          _writer.send("q1", answer);
    }
  }
```

And finally change the MessageReceiverImpl:

```
public class MessageReceiverImp extends java.rmi.server.UnicastRemoteObject implements MessageReceiver {
  MessageReceiverImpl() throws java.rmi.RemoteException { }
  public String message(String str) throws java.rmi.RemoteException {
    return "Got your message: " + str;
  }
}
```

סעיף ד (8 נקודות)

I	Yes
II	Yes/No
III	yes
IV	yes

שאלה 4 (10 נקודות)

סעיף א (6 נקודות)

```
CREATE TABLE PlaneModels (
  Model varchar(20) PRIMARY KEY,
```

```
   TechnicalSpec varchar(10000)
   Capacity integer)

CREATE TABLE Planes (
    ID integer PRIMARY KEY,
    NextMaintenancePlanned Date,
    BusinessClassCapacity integer,    -- Could also be put in the Flights or in the PlaneModels table
    Model varchar(20) FOREIGN KEY REFERENCES PlaneModels(Model))

CREATE TABLE Flights (
   ID integer PRIMARY KEY,
   Destination varchar(100),
   Terminal varchar(20),
   ExitGate varchar(20),
   ExitDate date,
   ExitTime time,
   PilotName varchar(100),
   PlaneId integer FOREIGN KEY REFERENCES Planes(ID))
```

Remarks:

- Different pilots can fly the same plane. Therefore the pilot name is located in the Flights table and not in the Planes table.
- The planned maintenance is specific to each plane instance. It does not depend on each flight. Therefore it is in the Planes table.
- The business class capacity generally is adjusted for each flight within a maximum that depends on the plane model. So the most likely setting would be to have the field in the Flights table and a maximum value in the PlaneModels table.

סעיף ב (4 נקודות)

```
Select Flights.ID, Planes.BusinessClassCapacity
From Flights JOIN Planes on Flights.PlaneId = Planes.Id
Order by Flights.destination asc
```