גיליון תשובות

מספר נבחן:_____

שאלה 1 (30 נקודות)

ראשית, מספר הערות על מאפייני משחק הקלפים (מטאפורה שְחוּקָה על החיים, הקורס, והמבחן) אשר פורטו בשאלה זו:
משחק מרתק המשלב הנאה. ראו מקרה הרב יהודה-אריה מודנה, מגדולי המלומדים היהודיים בשלהי תקופת הרנסס, אשר לצד חיבוריו הרבים, לא הניח ידו גם מחפיסת הקלפים.
זריזות ידיים. כפי שהדגים זאת בחייו, דוק הולידיי האגדי - רופא שיניים לעת מצוא, אקדוחן, שלפן, שתיין, שחפן, ובעיקר קלפן (לתוספת עיון, ראו המערבון הקלאסי Doc).
בהירות מחשבה וקורט של מזל. לעיתים, אגב, ההפך הוא הנכון.

סעיף א (4 נקודות)

@INV: 0 <_cards.size() < 66 &&
_cards.first() == 5 &&
for each *card* in *cards*: legalCord(*card*)

המחלקה בטוחה תחת כל חישוב מקבילי, כי האינווריאנטה נשמרת:
- כל השדות private.
- כל המתודות הניגשות למצב הפנימי (= השדות של המחלקה) מסונכרנות.

סעיף ב (2 נקודות)

@PRE: legalCard(card)
@POST: _cards.size() == @PRE(_cards.size()) + 1 && _cards.last() == card

סעיף ג (8 נקודות)

```
class CardTable {
...
  public synchronized int addCard(Integer card) throws WrongCardValueException {
    if (!legalCard(card))
      throw new WrongCardValueException(card);
    if (Math.abs(card - _cards.lastElement()) > 3)
      wait(10000);
    int ret = Math.abs(card - _cards.lastElement());
    _cards.add(card);
    notifyAll();
    return ret;
  }
...
}
```

הערה: למי שלא הכיר את החתימה wait(long time), ניתן היה, לחלופין, להפעיל את מנגנון sleep/interrupt על הת'רד.

סעיף ד (2 נקודות)

הקלף בראש השולחן 5, לשחקן אחד קלף 10 ולשני קלף 1.

סעיף ה (12 נקודות)

```
class Dealer implements Runnable{
  private CardTable _table;
  private PlayerDeadlockController _deadlockController;

  Dealer(CardTable table, PlayerDeadlockController deadlockController) {
    _table = table;
    _deadlockController = deadlockController;
  }
  public void run() {
    while (!_deadlockController.isEndOfTheGame()) {
      try {
        _deadlockController.waitForDeadlock();
      catch(InterruptedException e) {
        return;
      }
       _table.reset();
    } // while
  }
}

class PlayerDeadlockController {
  private int _iPlayers, int _iWaitedPlayers;

  PlayerDeadlockController(int iPlayers) {
    _iPlayers = iPlayers;
  }
  public synchronized void incWaitedPlayer() {
    _iWaitedPlayers++;
    if (_iWaitedPlayers == _iPlayers)
      notifyAll();
  }
  public synchronized void decWaitedPlayer() {
    _iWaitedPlayers--;
    if (_iWaitedPlayers == _iPlayers)
```

```
        notifyAll();
    }
    public synchronized void decPlayer() {
        _iPlayers--;
    }
    public synchronized void waitForDeadlock() throws InterruptedException {
        while (_iWaitedPlayers < _iPlayers)
            wait();
    }
    public synchronized boolean isEndOfTheGame() {
        return _iPlayers == 0;
    }
}

class CardTable {
    private Stack<Integer> _cards;
    PlayerDeadlockController _deadlockController;

    CardTable(PlayerDeadlockController deadlockController) {
        _cards = new Stack<Integer>();
        _cards.add(card);
        _deadlockController = deadlockController;
    }
    public synchronized int addCard(Integer card) throws WrongCardValueException, InterruptedException {
        if (!legalCard(card))
            throw new WrongCardValueException(card);
        while (!checkPreCond(card)) {
            _deadlockController.incWaitedPlayer();
            wait();
            _deadlockController.decWaitedPlayer();
        }
        int ret = Math.abs(card - _cards.lastElement());
        _cards.add(card);
        notifyAll();
        return ret;
    }
    private synchronized boolean checkPreCond(Integer card)) {
        return Math.abs(card - _cards.lastElement()) =< 4;
    }
    public synchronized void reset() {
        _cards.clear();
        _cards.add(Game.getRandomCard());
        notifyAll();
    }
```

```
}

class Player implements Runnable {
  private CardTable _table;
  private Queue<Integer> _cards;
  private int _debt;
  PlayerDeadlockController _deadlockController;
  Player(CardTable table, Queue<Integer> cards, PlayerDeadlockController deadlockController) {
    _table = table;   _cards = cards;   _debt = 0; _deadlockController = deadlockController;
  }
  public void run() {
    Random ran = new Random();
    while (!_cards.isEmpty() && !Thread.currentThread().isInterrupted()) {
      Integer card = _cards.remove();
      try {
        _debt += _table.addCard(card);
        Thread.sleep(rand.nextInt(1000));  // Wait a random delay between 0 and 1 second
      } catch (InterruptedException e) {
        break;
      }
    }// while
    _deadlockController.decPlayer();
  }
  public int getDebtPoints() { return _debt; }
}

class Game {
  public static Integer getRandomCard() {
    Random rand = new Random();
    return rand.nextInt(10)+1;
  }
  private static Queue<Integer> createCards(int size) {
    Queue<Integer> ret = new LinkedList<Integer>();
    for (int i=0; i<size; i++)
      ret.add(getRandomCard());
    return ret;
  }
  public static void main(String[] args) throws InterruptedException, WrongCardValueException {
    PlayerDeadlockController deadlockController = new PlayerDeadlockController(2);
    CardTable table = new CardTable(deadlockController);
    Player player1 = new Player(table, createCards(32), deadlockController);
    Player player2 = new Player(table, createCards(32), deadlockController);
    Dealer dealer = new Dealer(table, deadlockController);
    Thread t1 = new Thread(player1);
```

```
    Thread t2 = new Thread(player2);
    Thread t3 = new Thread(dealer);
    t1.start();
    t2.start();
    t3.start();
    t1.join();
    t2.join();
    t3.join();
    if (player1.getDebtPoints() == player2.getDebtPoints())
      System.out.println("No winner for this game...");
    else if (player1.getDebtPoints() < player2.getDebtPoints())
      System.out.println("Player1 won this game");
    else
      System.out.println("Player2 won this game");
  }
}
```

סעיף ו (2 נקודות)

@INV: 0 <_cards.size() < 66 &&
for each ($card_i$, $card_{i+1}$) in *cards*: abs($card_i$ - $card_{i+1}$) < 4

שאלה 2 (30 נקודות)

סעיף א (3 נקודות)

ColorPrinter - print
Printing is not supported

סעיף ב (5 נקודות)

ColorPrinter - print
ColorImage - printMe
ColorPrinter printing ColorImage
ColorPrinter – print
BWImage - printMe
ColorPrinter printing BWImage

סעיף ג (5 נקודות)

הפלט זהה לפלט המקורי

ColorPrinter - print
ColorImage - printMe
ColorPrinter printing ColorImage

סעיף ד (12 נקודות)

המחסנית:

ערך: 2000 גודל מצביע: 4. מצביע לאובייקט **ColorImage**
ערך: 1000 גודל מצביע: 4. מצביע ל vtable של P.
כתובת חזרה ל main

הערימה:

2000

3000	X	X	X								
vtable גודל 4. מצביע ל	גודל 12: עבור תשעה ערכי char ו- alignment.										

סעיף ה (5 נקודות)

להוסיף

```
virtual ~Image() {}
```

ה **destructor** שמתבצע בקריאה **delete colorImg** הוא של **Image** בהתאם לסוג הסטטי של האובייקט. על מנת שיתבצע **binding** בזמן ריצה ל **colorImage~** צריך שימוש ב **virtual**

שאלה 3 (30 נקודות)

<u>סעיף א</u>:

We see in the code the following calls:
On the server side:

```
Naming.rebind( "//132.24.56.8:2002/Vote", voting);
```

On the client side:

```
Voting voting = (Voting)Naming.lookup("//132.24.56.8:2002/Vote");
```

The Java Naming class is a client of the RMI name server (by default rmiregistry).
Both the client and the server assume they interact with the same name server, located on host at IP 132.24.56.8 on port 2002.

<u>סעיף ב</u>:

```
Set<String> parties = voting.getParties();
```

2 processes are running. Each process runs in its own virtual memory (possibly on two different hosts). The answer must, therefore, explain how many copies of the object are made in each process and in which form.

The call to voting.getParties() is performed by the client. When this happens, the voting stub sends a request to the voting skel on the server. On the server process, the skel passes the request to the implementation. The implementation executes this line:

```
public synchronized  Set<String> getParties() throws java.rmi.RemoteException {
    return _mapParty2Votes.keySet();   // return the set of keys in the map as a set of strings
}
```

This method computes a set of strings (by extracting the keys from the TreeMap data-structure).

How can we know whether this computation (keySet()) actually performs a copy of the keys stored in the TreeMap or returns a reference to an internal Set managed inside the TreeMap object? If we had access to the JavaDoc, we would read this:

public Set<K> **keySet**()

> Returns a Set view of the keys contained in this map. The set's iterator will return the keys in ascending order. The map is backed by this `TreeMap` instance, so changes to this map are reflected in the Set, and vice-versa.

Naturally, in the exam, we cannot know for sure – but we just write our assumption (keySet() performs one copy or does not perform one copy of the list of parties) – both answers were acceptable.

The reference to this set of strings is then passed to the skel. Since we are operating within a single Java process, the return value is passed by reference, and there is no copy performed (from the call of VotingImpl.getParties() to the skel).

The skel then serializes the value into its socket. This serialization is a form of copy. It sends the serialized form of the Set into the socket. (Serialization in RMI is similar in function to a call to the Copy Constructor in C++.)

NOTE: When a return value or a parameter are passed in RMI, there are 3 options:

1. The object is copied by value (if the type is serializable)
2. It is passed by reference if the type extends RemoteObject
3. The code does not compile (if the type is not serializable and does not extend RemoteObject).

Since we see code that runs, we can assume during the exam that option (3) is not the right one. Do we have any reason to suspect that Set<String> extends RemoteObject? None. So we must conclude that Set<String> is serializable (which is indeed the case).

The client stub then de-serializes the data it gets from the socket, and constructs a copy of the list of strings in the memory of the Client process. This is a second copy of the data structure. (De-serialization operates like a constructor that receives a byte buffer parameter.)

The stub method getParties() eventually returns the constructed value to the caller, which stores it into the variable parties. Since we are working inside a single Java process, this is performed as a return value by reference and no copy is happening.

Overall, the call caused 2 copies of the set of string data-structure: one by the skel serializing the data in the server process, and one by the stub de-serializing the data in the client process.

סעיף ג:

We start from the vote() method which was graciously given next to the method to implement and copy it almost line by line. We also look at the code of the processMessage of the VotingProtocol class and how the reply to the message GET_PARTIES is serialized on the server side.

Our task in the stub:

1. Send the appropriate request to the VotingProtocol in the right format, and with the right framing.
2. Read the answer sent by the VotingProtocol
3. De-serialize the answer and construct the right Java object from the message we get back.
4. Detect any error that could happen – because of communication errors on the socket or because the VotingProtocol signaled an error and throw an exception if we get such an error.

```
  public void vote(String party) throws RemoteException {
    try {
      Socket socket = new Socket(_skelHost,_skelPort);
      String msg = "VOTE " + party + "\n";
      socket.getOutputStream().write(msg.getBytes("UTF-8"));
      BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream(),"UTF-8"));
      String ans = in.readLine();
      msg = "BYE\n";
      socket.getOutputStream().write(msg.getBytes("UTF-8"));
      socket.close();
    } catch (Exception e) {
      throw new RemoteException(e.toString());
    }
    if  (!ans.equals("SUCCESS"))
      throw new RemoteException("Vote failed");
  }
}

  public Set<String> getParties() throws RemoteException {
    try {
      Socket socket = new Socket(_skelHost,_skelPort);
```

```
    //@@1 Send the right request – do not forget the "\n" which is the framing of the request
    String msg = " GET_PARTIES\n";
    socket.getOutputStream().write(msg.getBytes("UTF-8"));

    // @@2 Read the answer from the server – use the readLine() method according to the framing used
    BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream(),"UTF-8"));
    String ans = in.readLine();

    // @@3 Send an end of invocation signal to the protocol and close the connection
    msg = "BYE\n";
    socket.getOutputStream().write(msg.getBytes("UTF-8"));
    socket.close();
  } catch (Exception e) {
    // @@4: If there was any I/O error when using the socket – throw.
    throw new RemoteException(e.toString());
  }
  // @@5 Verify that the call succeeded on the server side – else throw.
  if  (ans.equals("FAIL"))
    throw new RemoteException("Call failed on server");
   }

   // @@6 De-serialize the answer into a Set<String>.
   // @@    The answer was encoded in the Protocol as a list of strings with a '#' delimiter.
   TreeSet<String> ret;
   StringTokenizer st = new StringTokenizer(ans, "#");
   while (st.hasMoreTokens()) {
     ret.add(st.nextToken());
   }
   return res;
 }
```

<u>סעיף ד</u>:

- כמה מסופי הצבעה (**VotingClient**) יכולים להיות מחוברים ל **VotingImpl**?

There is no logical limitation on the number of clients that can be connected to the reactor. The system limitations come from the number of input channels the selector on the reactor can accept (this is commonly limited to about 1000 channels) and from the RAM usage associated to each connection on the reactor side (there is one attachment with some buffers associated to each open connection).

- כמה בקשות הצבעה יכולות להיות מטופלות במקביל ב **VotingProtocol**?

When a vote request is processed, a thread of the Executor on the reactor side is allocated. There are 10 threads in the reactor + 1 thread for the reactor itself, as indicated by this code:

```
int poolSize = 10;
new Reactor(port, poolSize, protocolMaker, tokenizerMaker).start();
```

Therefore, there can be up to 10 active vote requests handled at a single time.

- כמה בקשות הצבעה יכולות להתבצע במקביל במתודה vote במחלקה VotingImpl?

The method VotingImpl.vote() is synchronized. Therefore, there can be only one thread that executes it at any single time.

סעיף ה:

We changed the signature of the vote() method in the interface.
This does not have any impact on the Voting_Skel class.
It has an impact on VotingProtocol, Voting_Stub and VotingImpl.

In the Stub, we must add a parameter to the method and pass the value of this parameter in the serialization of the call:

```
public class Voting_Stub implements Voting {
  public void vote(String party, long userID) throws RemoteException {
    try {
      ....
      // @@ Add the userId parameter to the request sent to the protocol – make sure the request
      // @@ is easily parsable – we assume "#" does not appear in the name of parties.
      String msg = "VOTE " + party + "#" + userId + "\n";
      .... No more changes ....
}
```

In the protocol, we must de-serialize the call by parsing what we encoded in the stub:

```
public class VotingProtocol implements AsyncServerProtocol {
  public String processMessage(String msg) {
    if (msg.startsWith("VOTE ")) {
      try {
        // @@ Get both arguments after the name of the method then split them
        String params = msg.substring(5);
        String party = params.split("#")[0];
        long userId = Long.parseLong(params.split("#")[1].trim());
        _voting.vote(party, userId);
        return "SUCCESS";
```

```
      } catch (RemoteException e) {
        return "FAIL";
      } ...
  }
```

In the implementation, we must:

- Initialize the database of registered voters in the constructor (and add a member to store it)
- Add a new parameter to the vote method
- Test the 2 requested conditions in the vote method:
  - User is registered
  - User has not already voted
- Remember that the user has voted so that he cannot vote again

```
public class VotingImpl extends java.rmi.server.UnicastRemoteObject implements Voting {
  private final Map<String,Long> _mapParty2Votes;
  // @@ Add a new list of registered voters and voters who have already voted
  private final List<Long> _registeredVoters;
  private List<Long> _votedVoters;
  // @@ Add a parameter to the constructor to get the list of registered voters
  VotingImpl(String partiesDatafile, String votersDatafile) throws java.rmi.RemoteException {
    _mapParty2Votes = new TreeMap<String,Long>();
   // read line by line the list of parties running in this election from datafile
    try {
      BufferedReader in = new BufferedReader(new InputStreamReader(
            new FileInputStream(partiesDatafile),"UTF-8"));
      String party = null;
      while ((party = in.readLine()) != null)
        _mapParty2Votes.put(party,0L);
    } catch (Exception e) {
      throw new java.rmi.RemoteException(e.toString());
    }

    // @@ Read the registered voters
    _votedVoters = new List<Long>();
    _registeredVoters = new List<Long>();
   // read line by line the list of parties running in this election from datafile
    try {
      BufferedReader in = new BufferedReader(new InputStreamReader(
                new FileInputStream(votersDatafile),"UTF-8"));
      String voter = null;
      while ((voter = in.readLine()) != null)
        _registeredVoters.add(Long.parseLong(voter.trim()));
    } catch (Exception e) {
      throw new java.rmi.RemoteException(e.toString());
```

```
    }
  }
  // @@ Add a userId parameter
  public synchronized void vote(String party, long userId) throws java.rmi.RemoteException {
    // @@ Test that user is a registered voter
    if (!_registeredVoters.contains(userId))
        throw new java.rmi.RemoteException("User "+userId+" is not registered");
    // @@ 2nd condition: user has not already voted
    if (_votedVoters.contains(userId))
        throw new java.rmi.RemoteException("User "+userId+" has already voted");
    Long votes = _mapParty2Votes.get(party);
    if (votes == null)
      throw new java.rmi.RemoteException("Party "+party+" is not running in this election");
    else {
        _mapParty2Votes.put(party,votes+1);
        // @@ Remember that user voted
        _votedVoters.add(userId);
    }
  }
}
```

שאלה 4 (10 נקודות)

Required tables:

```
CREATE TABLE VOTER (
    IdNumber varchar(9) PRIMARY KEY,
    KalpiNumber integer FOREIGN KEY References Kalpi.KalpiNumber,
    HasVoted Boolean)

CREATE TABLE KALPI (
    KalpiNumber integer PRIMARY KEY,
    City varchar(200),
    OpeningHour time)

CREATE TABLE PARTIES (
    PartyName varchar(200),
    VotesReceived integer)
```

Query:

```
SELECT IdNumber from VOTERS
WHERE KalpiNumber = 17 and HasVoted = false
```