אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

רשמו תשובותיכם בגיליון התשובות בלבד
תשובות מחוץ לגיליון לא יבדקו.

בהצלחה!

תאריך הבחינה: 17.4.2009
שם המורה: ד"ר מיכאל אלחדד
ד"ר מני אדלר
מר אוריאל ברגיג
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשס"ט
סמסטר: א'
מועד: ב'
משך הבחינה: שלש שעות
חומר עזר: אסור

שאלה 1 (30 נקודות)

בשאלה זו נעסוק בגרסה האחרונה שבמשחק הקלפים שהוצג במועד א'. כזכור, חוקי המשחק מוגדרים כדלהלן:

- הקלפים במשחק מאופיינים על ידי מספר שלם בתחום [1-10].
- במשחק משתתפים שני מתחרים.
- כל אחד משני המתחרים מקבל חפיסת קלפים עם 32 קלפים אקראיים.
- לכל מתחרה יש מונה של 'נקודות חובה' המאותחל לאפס.
- על השולחן מונחת ערימה של קלפים. בתחילת המשחק מכילה ערימה זו קלף אחד עם ערך 5.
- מכאן ואילך לוקח כל מתחרה, בקצב שלו (כלומר, המשחק לא מתנהל תור-תור), את הקלף העליון מחפיסתו: אם הקלף שבידו תואם את הקלף הנמצא בראש הערימה בשולחן (= ההפרש המוחלט ביניהם אינו עולה על 3) הוא מניחו על הערימה, אחרת הוא ממתין (על ידי ביצוע wait על הערימה בשולחן) לשינוי הערך של הקלף בראש הערימה.
- ההפרש המוחלט, בין ערך הקלף המונח על הלוח בשולחן לבין זה של המתחרה, מתווסף לנקודות החובה שלו.
- המשחק מסתיים כאשר הונחו כל קלפי השחקנים על הלוח.
- מנצח המשחק הינו השחקן בעל מספר נקודות החובה הנמוך יותר.

משחק שכזה עלול להיקלע לחבק (deadlock): אם לשני השחקנים יש קלף שאינו תואם לקלף בראש הערימה, הם ימתינו לנצח לשינוי בערימת הקלפים. כדי לפתור תרחיש מסוכן זה של deadlock, הוספנו למערכת אובייקט אקטיבי (ת'רד) Dealer, המזהה את החבק, מנקה את השולחן מהקלפים שעליו, מניח קלף אקראי כל שהוא, ומעיר את השחקנים הממתינים.
הקוד של המערכת, כפי שניתן בפתרון של מועד א', ניתן בנספח 1 של המבחן. <u>מרבית הקוד אינה נדרשת כלל על מנת לענות על השאלה (אנו נותנים אותו רק לשם עזרה אם אתם נתקלים בבעיה, או לא מבינים חלקים במערכת שתוארה)</u>.

בהרצאות בכיתה למדנו על תרחיש בעייתי נוסף – LiveLock.
בתרחיש ה DeadLock ת'רדים למצב blocked (עקב המתנה למוניטור של אובייקט לפני קטע קוד מסונכרן, או עקב המתנה לשינויים באובייקט) ממנו לא יצאו לנצח.
בתרחיש ה LiveLock, לעומת זאת, הת'רדים במערכת עובדים (נכנסים מידי פעם למצב blocked אך גם יוצאים ממנו) אולם התוכנית בכללה לא מצליחה להתקדם.

נגדיר את 'התקדמות התוכנית' במקרה שלנו כהקטנת כמות הקלפים הנמצאת בידי השחקנים.

א. הראו כיצד עלולה התוכנית שתיארנו עד כה להיקלע ל LiveLock (כלומר, ציינו תרחיש בו אין אמנם deadlock בעקבות פעולת ה Dealer אולם כמות הקלפים בידי השחקנים אינה משתנית).
<u>הערה</u>: אין צורך לעיין בקוד התוכנית (המובא בנספח 1 למבחן), מומלץ מאוד להפעיל הגיון פשוט, על המערכת שתוארה במילים. (בכל מקרה, ניתן להתעלם מהקריאה ל printCards במתודת ה main בה נעסוק בסעיפים ג-ד).
[5 נקודות]

ב. כדי למנוע תרחיש LiveLock שכזה, עדכנו את ה התוכנית באופן הבא: לאחר עשרה ניסיונות של התרת ה deadlock בהם לא קטן מספר הקלפים שביד השחקנים, מפסיק ה Dealer את פעילותם של השחקנים, והמנצח הוא השחקן בעל המספר המינימאלי של נקודות החובה.
[10 נקודות]

<u>ניתן לענות על סעיפים ג-ד, גם אם ויתרתם על א-ב</u>. כמו כן ניתן להתייחס בסעיפים ג-ד לקוד המקורי בנספח השאלה.

כדי לעקוב אחר מהלך המשחק, נוספה למחלקה CardTable מתודה מסונכרנת printCards המדפיסה למסך את תוכן ערימת הקלפים. במערכת המוצגת בנספח לשאלה, לדוגמא, מתבצעת קריאה למתודה printCards על ידי ת'רד הראשי במתודה main של המחלקה Game, לאחר הפעלת ת'רדים.

ג. המתודה printCards נועלת את לוח הקלפים <u>לכל משך ביצוע ההדפסה</u> (כלומר, כל ההדפסה מתבצעת תחת סינכרון על cardTable). הסבירו מדוע נדרשת נעילה זו, על ידי תאור תרחיש בו ביטול סנכרון המתודה printCards, גורם לפלט שאינו מקיים את האינווריאנטה של ערימת הקלפים (האינווריאנטה, כזכור, הינה: כל קלף תואם את הקלף הקודם לו בערימה).
[5 נקודות]

ד. ממשו את המתודה printCards על פי עקרונות fail fast/optimistic try and fail שנלמדו בהקשר של version iterator בהרצאות.
אין חובה לממש את ה version iterator בדיוק כפי שתואר בכיתה, מספיק ליצור מנגנון פשוט במחלקה CardTable, על בסיס אותו עיקרון, התומך בזיהוי של שינו כלשהוא שנעשה על ידי ת'רדים אחרים בתוכן של _cards.
[10 נקודות]

שאלה 2 (30 נקודות)

נתון הקוד הבא, הקוד עובר קומפילציה ויוצר קובץ להרצה. בהמשך שאלות המתייחסות לקוד

```
class B {
private:
  int *m_pData;
public:
  B(int data):m_pData(new int(data)) {}
  B& operator=(const B& other) {
    clean();
    copy(other);
    return *this;
  }
  virtual const int getData() const {
```

```
    return *m_pData;
  }
  virtual ~B() {
    clean();
  }
private:
  void copy(const B& other) {
    m_pData = other.m_pData;
  }
  void clean() {
    if (m_pData) delete m_pData;
      m_pData = 0;
  }
};

class D : public B {
private:
  int *m_pData;
public:
  D::D() : B(1), m_pData (0) {}
  virtual const int getData() const {
    // סעיף ה –תמונת הזיכרון במקום הזה
    return -1;
  }
};

void Q2()
{
  B b1(1);
  B b2(2);
  b2 = b1;
}

void Q3() {
  B b1(1);
  b1 = b1;
  int data = b1.getData();
}

void Q4_func2(B* b)
{
  int i = 5;
  // סעיף ד –תמונת הזיכרון במקום הזה
}
```

```
void Q4()
{
  B b1(1);
  B *pb1 = &b1;
  Q4_func2(pb1);
}

void Q5()
{
  D d;
  d. getData();
}

void main()
{
  // כאן מתבצעת קריאה יחידה לאחת מהשיטות בהתאם לסעיפים
}
```

א. קריאה לשיטה Q2 בלבד מה main גורם לפעולה לא תקינה על ה heap של התהליך. הסבירו בקצרה.
[3 נקודות]

ב. קריאה לשיטה Q3 בלבד מה main גורמת לקריסה של התהליך. הוסיפו שורת קוד אחת ב B& operator=(const B& other) על מנת למנוע קריסה זו.
[5 נקודות]

ג. הוסיפו בנאי מעתיק, copy constructor, למחלקה B
[2 נקודות]

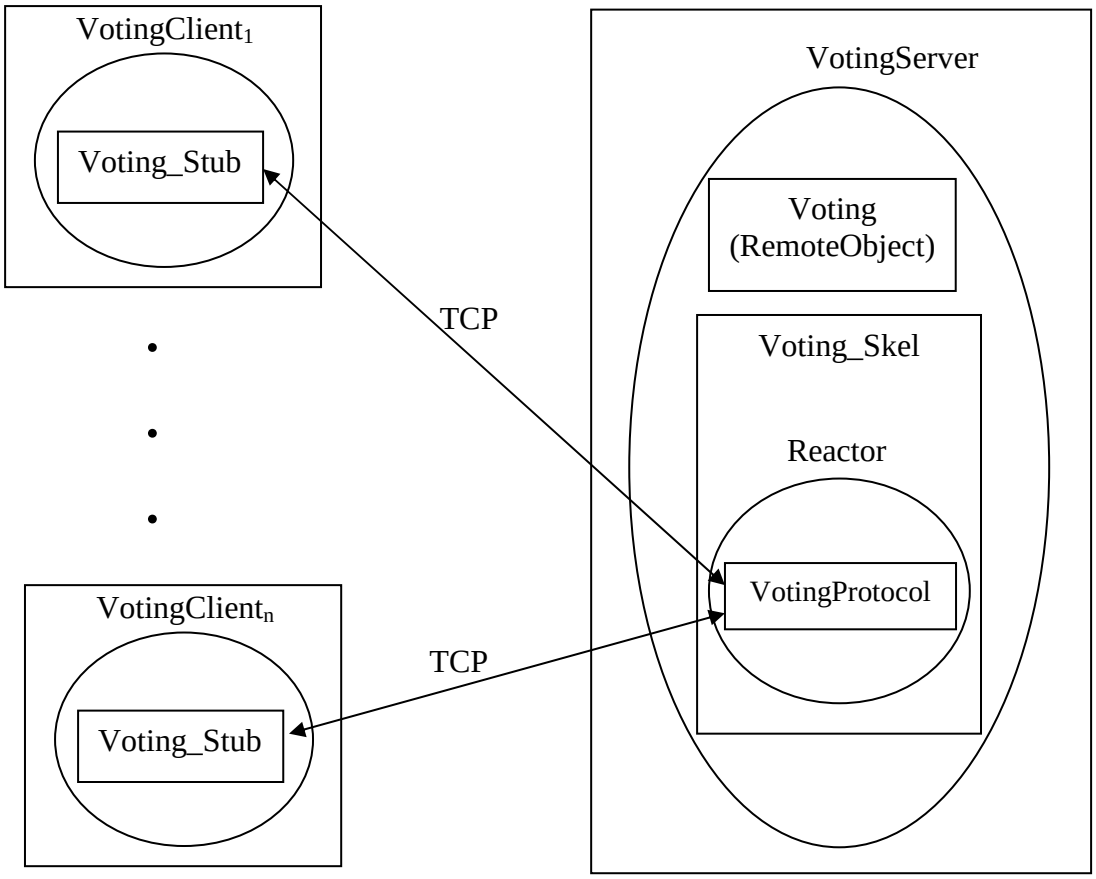
בסעיפים הבאים יש לצייר את תמונת הזיכרון של התהליך הכוללת את הערכים הנמצאים במחסנית, בערימה, מצביעי vtable אם יש וה- vtables עצמם אם יש. יש לכלול באיור כתובת הזיכרון בא מתחיל כל ערך בהינתן שגודלו של int הוא 4 וגודלו של מצביע הוא 4. לשם האחידות נקבע כי כתובות בערימה הם בתחום 1000 - 3000 וכי המחסנית מתחילה בכתובת 5000.

ד. ציירו תמונת הזיכרון בהנחה שה- main קורא ל Q4 בלבד, ו Q4 קרא ל Q4_func2. שימו לב, בקוד של Q4_func2 מופיע הערה עם מראה מקום שסעיף זה מתייחס אליו.
[10 נקודות]

ה. ציירו תמונת הזיכרון בהנחה שה- main קורא ל Q5 בלבד, ו Q5 קרא ל getData של המחלקה D. שימו לב, בקוד getData של המחלקה D מופיע הערה עם מראה מקום שסעיף זה מתייחס אליו.
[10 נקודות]

שאלה 3 (30 נקודות)

בשאלה זו נמשיך לעסוק במערכת ההצבעה דרך מסופי מחשב, בה עסק מועד א'.
להלן תאור המערכת (זו אותה מערכת בדיוק ממועד א):



תהליך השרת (VotingServer) מגדיר RemoteObject הממש את הממשק Voting התומך בהצבעה למפלגה מסוימת, ובמתודה המחזירה את רשימת המפלגות.

ה Skel של ה RemoteObject (Voting_Skel) מגדיר אובייקט מסוג Reactor המטפל בבקשות הלקוחות על פי VotingProtocol. פרוטוקול זה מקבל בקשות (כמערך של בתים) מערוץ קלט של קשר TCP (SocketChannel), מבצע את הפעולה על אובייקט ה Voting בתהליך השרת הממש פעולות אלו, ומחזיר תשובה ללקוח לערוץ הפלט של קשר ה TCP.

תהליך הלקוח (VotingClient) מכיל את אובייקט ה Voting ומבצע עליו פעולות של קבלת רשימת מפלגות והצבעה למפלגה מסוימת, על פי דרישות המשתמש.
ראינו כי אובייקט ה Voting זה הינו למעשה עותק של Voting_Stub. אובייקט זה ממש את ממשק ה Voting על ידי התחברות ל Reactor של ה Voting_Skel בקשר TCP (Socket), ושליחת בקשות (כמערך של בתים) על פי הפרוטוקול של ה Reactor (VotingProtocol).

הקוד של המערכת, כפי שניתן בפתרון של מועד א', מובא בנספח 2 למבחן. מרבית הקוד אינו נדרש כלל על מנת לענות על השאלה (אנו נותנים אותו רק לשם עזרה אם אתם נתקלים בבעיה, או לא מבינים חלקים במערכת שתוארה).

בעקבות הצלחת המערכת בבחירות בארץ, החליטה נסיכות ליכטנשטיין לאמץ את המערכת עבור הבחירות הממשמשות ובאות.
בנסיכות ליכטנשטיין ישנן רק עשר קלפיות. מנתחי המערכות בממלכה, טוענים כי בשל עובדה זו, במקום להגדיר Reactor ב Voting_Skel אפשר להסתפק ב MultiServer - אשר פותח על ידי מתרגלי הקורס 'תכנות מערכות' באוניברסיטת בן גוריון - המגדיר ת'רד נפרד לכל לקוח.

א. חוו דעתכם בקצרה על הצעה זו , בהבטים הבאים:
- שרות הוגן ללקוחות
- תמיכה בהתרחבות עתידית של הממלכה
- ניצולת CPU
- כמות המשאבים הנדרשת

[8 נקודות]

ב. השלימו את השורה החסרה במחלקה Voting_Skel כך שמודל השרת יתנהל על פי הצעת מנתחי המערכות של נסיכות ליכטנשטיין. [4 נקודות]

```
public class Voting_Skel {
  Voting_Skel(Voting voting) throws Exception {
    int port = 1984;
    @TODO: complete this line
  }
}
```

חומר עזר: חתימת הבנאי של המחלקה **MultipleClientProtocolServer** המממשת את הממשק **Runnable**.
כזכור, מחלקה זו נלמדה בתרגול מספר 10, ובה השתמשתם במשימת התכנות השלישית והאחרונה.

```
MultipleClientProtocolServer(int port, AsyncServerProtocol  p)
```

סעיפים ג-ה מתייחסים לקוד המקורי, וניתן לענות עליהם גם אם ויתרתם על א-ב.

השמועה על מערכת ההצבעה המתקדמת הגיעה למדינה בעלת שלטון טוטליטרי-רודני (שאת שמה לא ניתן, מטעמי בטיחות כמובן, למסור בשלב זה). במדינות מסוג זה, כידוע, נוהגים לקיים באדיקות את טקס הבחירות, אף כי אין להצבעה של האזרחים כל השפעה על התוצאה הסופית.
קבוצת מדענים איראניים, המועסקים כיועצים חיצוניים ברודנות הנ"ל, הבחינה בעובדה, שהן ההודעה עם הבקשה לקבלת רשימת המפלגות, והן ההודעה עם הבקשה להצביע למפלגה, נשלחות בקשר TCP. מאחר ואין כל משמעות להצבעה של האזרח, למעט העובדה שהוא 'הצביע', הציעו המדענים האיראניים, לשלוח את ההודעה עם הבקשה להצביע ב UDP במקום ב TCP.

ג. הסבירו מדוע המליצו המדענים לעבור ל UDP:
- האם זה חוסך פעולות תקשורת?
- האם זה דורש פחות זכרון מהלקוח והשרת?

נמקו ופרטו תשובותיכם.

[6 נקודות]

כדי לממש את המערכת שהוצעה על ידי המדענים האיראניים, יש לבצע שני שינויים במערכת:

1. שינוי המתודה vote במחלקה Voting_Stub כך שבקשת ההצבעה תשלח ב UDP מ DatagramSocket, במקום הכתיבה לערוץ הפלט של ה Socket במימוש הנוכחי המבוסס על TCP. בניגוד למימוש הקיים, לאחר שליחת בקשת ההצבעה הלקוח אינו מחכה לתשובה מהשרת.
2. עדכון ה **ConnectionAcceptror** ב **Reactor**, כך שבנוסף לרישום ה **SocketChannel** ב **Selector** עם **ConnectionHandler**, יתבצע בנוסף רישום של **DatagramChannel** עם אותו **ConnectionHandler**.

DatagramChannel הינה מחלקה ב **java.nio**. **DatagramChannel** מחובר ל host ול port של לקוח. בדומה ל **SocketChannel** הוא מרחיב את **SelectableChannel** תוך מימוש פעולות קריאה וכתיבה ל **Channel** על ידי המתודות **read, write**. אולם בניגוד ל **SocketChannel** שקורא וכותב מידע מקשר TCP, הקריאה והכתיבה הממומשות ב **DatagramChannel** הינן על בסיס UDP. הקריאה והכתיבה ב **DatagramChannel** צריכה להיות כמובן non-blocking.

ד. השלימו את עדכון המתודה **vote** ב **Voting_Stub** על פי תאור דרישה 1 למעלה. למחלקה נוספו שדות – הגדרת השדות ואיתחולם מסומנים בקו תחתון. . הקוד החסר מצויין על ידי @@.
[6 נקודות]

```
public class Voting_Stub implements Voting {
  String  _skelHost;
  int _skelPort;
  InetAddress _skelAddress;
  DatagramSocket _datagramSocket ;

  Voting_Stub(String skelHost,int skelPort) throws RemoteException {
    _skelHost = skelHost; _skelPort = skelPort;
    _datagramSocket = new DatagramSocket();
    _ skelAddress = InetAddress.getByName(_skelHost);
  }
  …
   public void vote(String party, long userId) throws RemoteException {
    try {
      String msg = "VOTE " + party + "#" + userId + "\n";
      @@TODO
    } catch (Exception e) {
      throw new RemoteException(e.toString());
    }
}
```

ה. עדכנו את המתודה **accept** ב **ConnectionAcceptor**, על פי דרישה 2 למעלה. הקוד שנוסף מסומן בקו תחתון, וזה החסר מצויין על ידי @@.
[6 נקודות]

```
public void accept() throws IOException {
  // Get a new channel for the connection request
  SocketChannel sChannel = _ssChannel.accept();
   // If serverSocketChannel is non-blocking, sChannel may be null
  if (sChannel != null) {
```

```
    sChannel.configureBlocking(false);
    SelectionKey key = sChannel.register(_data.getSelector(), 0);
    ConnectionHandler handler = ConnectionHandler.create(sChannel, _data, key);
    handler.switchToReadOnlyMode(); // set the handler to read only mode
    InetSocketAddress  clientAddress = new InetSocketAddress(
                                    sChannel.socket().getRemoteSocketAddress(),sChannel.socket().getPort());
    @@TODO
}
```

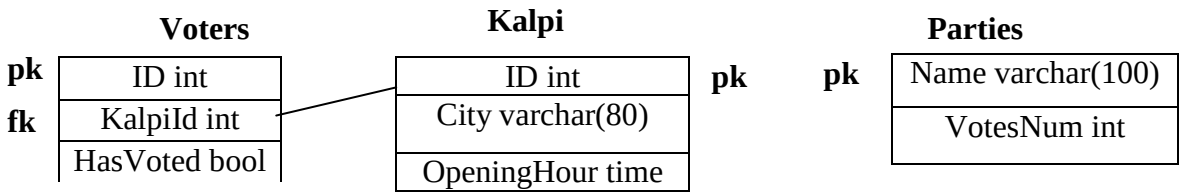
חומר עזר: מתודות נבחרות מה Java Doc של המחלקה **DatagramChannel**:

static DatagramChannel	**open**() Opens a datagram channel.
abstract DatagramChannel	**connect**(SocketAddress remote) Connects this channel's socket.

שאלה 4 (10 נקודות)

המערכת שנבנתה בסעיפים ג-ד בשאלה הקודמת הוצגה בפני העסקנים הבכירים במפלגת השלטון הרודני, והם אהבו כמובן את הרעיון. לשם השלמת 'הקונספט', נוספה מצידם דרישה: והיה ובקשת ההצבעה של הבוחר הגיעה בכל זאת לשרת, יש לשמור את פרטי ההצבעה שלו בבסיס הנתונים (לשם שימוש עתידי במידע חתרני זה).
במילים אחרות: במקום לשמור את מספר המצביעים לכל מפלגה, כפי שנעשה עד כה, יש לשמור כעת את המפלגה אליה הצביע כל בוחר.

להלן תאור מודל הנתונים ממועד א'.



א. עדכנו את המודל בהתאם לדרישה החדשה. [5 נקודות]

ב. כתבו שאילתה המחזירה את מספרי הזהות של הבוחרים שלא הצביעו למפלגת השלטון ' The Worms'. [5 נקודות]

נספחים

1. הקוד של משחק הקלפים מהפתרון למועד א' [עבור שאלה 1].

```
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();
  }

  public synchronized void print cards() {
      for (Integer card : _cards)
        System.out.println(card);
  }
}
```

```
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 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();
    table.printCards();
    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");
  }
}
```

2. הקוד של מערכת ההצבעה מהפתרון למועד א' [עבור שאלה 3].

```
public interface Voting extends java.rmi.Remote {
  Set<String> getParties() throws java.rmi.RemoteException;
  void vote(String party, long userID) throws java.rmi.RemoteException;
}
```

```
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);
    }
  }
}
```

```
public class VotingServer {
  public static void main(String[] args) {
    Voting voting = null;
    try {
      voting = new VotingImpl("parties.txt");
      Naming.rebind( "//132.24.56.8:2002/Vote", voting);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

```
public class VotingClient {
 public static void main(String[] args) {
  try {
    Voting voting = (Voting)Naming.lookup("//132.24.56.8:2002/Vote");
    Set<String> parties = voting.getParties();
    while (true) {
      System.out.println("Parties: " + parties);
      System.out.println("Please type the name of your favorite party, and press Enter");
      BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
      try {
         voting.vote(in.readLine());
      } catch (RemoteException e) {
          System.out.println(e.toString());
      }
```

```
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

```
public class Voting_Skel {
  Voting_Skel(Voting voting) throws Exception {
    ServerProtocolFactory protocolMaker = new ServerProtocolFactory() {
      public AsyncServerProtocol create() {
        return new VotingProtocol(voting);
      }
    };
    final Charset charset = Charset.forName("UTF-8");
    TokenizerFactory tokenizerMaker = new TokenizerFactory() {
      public StringMessageTokenizer create() {
        return new FixedSeparatorMessageTokenizer("\n",charset);
      }
    };
    int port = 1984;
    int poolSize = 10;
    new Reactor(port, poolSize, protocolMaker, tokenizerMaker).start();
  }
}
```

```
public class VotingProtocol implements AsyncServerProtocol {
   private boolean _shouldClose, _connectionTerminated;
   private Voting _voting;
   VotingProtocol(Voting voting) {
     _shouldClose = false;
     _connectionTerminated = false;
    _voting = voting;
  }
  public String processMessage(String msg) {
    if (_connectionTerminated)
      return null;
    if (isEnd(msg)) {
      _shouldClose = true;
      return "CLOSED";
    }
    if (msg.startsWith("VOTE ")) {
```

```
        try {
          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";
        }
      } else if (msg.startsWith("GET_PARTIES")) {
          try {
            Set<String> parties = _voting.getParties();
          StringBuilder sb = new StringBuilder();
          for (String party : parties) {
            sb.append(party);
            sb.append("#");
          }
          return sb.toString();
        } catch (RemoteException e) {
          return "FAIL";
        }
      }
      return "FAIL";
    }

    public boolean isEnd(String msg) {  return msg.equals("BYE");  }
    public boolean shouldClose() {   return _shouldClose;   }
    public void connectionTerminated() {   _connectionTerminated = true;   }
}
```

```
public class Voting_Stub implements Voting {
  String  _skelHost;
  int _skelPort;

  Voting_Stub(String skelHost,int skelPort) throws RemoteException {
    _skelHost = skelHost; _skelPort = skelPort;
  }

   public Set<String> getParties() throws RemoteException {
    try {
      Socket socket = new Socket(_skelHost,_skelPort);
      String msg = " GET_PARTIES\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("FAIL"))
        throw new RemoteException("Call failed on server”);
       }
       TreeSet<String> ret;
       StringTokenizer st = new StringTokenizer(ans, “#”);
       while (st.hasMoreTokens()) {
         ret.add(st.nextToken());
       }
       return ret;
    }

  public void vote(String party, long userId) throws RemoteException {
    try {
      Socket socket = new Socket(_skelHost,_skelPort);
      String msg = "VOTE " + party + “#” + userId + "\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");
  }
}
```