# אוניברסיטת בן-גוריון

## מדור בחינות

**מספר נבחן:_____**

רשמו תשובותיכם בגיליון התשובות בלבד
תשובות מחוץ לגיליון לא יבדקו.

**בהצלחה!**

תאריך הבחינה: 17.2.2006
שם המורה: ד"ר מיכאל אלחדד
מני אדלר
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשס"ו
סמסטר: א'
מועד: א'
משך הבחינה: שלש שעות
חומר עזר: אסור

## שאלה 1 (30 נקודות)

המחלקה Stock מייצגת מניה של חברה בשוק ההון. מניה מורכבת משם (_name), שער בסיסי (_base), מספר קונים (_ref), ומחיר (_price). מחיר המניה הוא סכום השער הבסיסי ומכפלת מספר הקונים ב 0.1.

```
class Stock  {
        private final String name_;
        final private double base_;
        private int ref_;
        private double price_;

        //@INV:  base_ >= 0 && ref_ >= 0 && price_ ==  (base_ + 0.1 * ref_)
        Stock(String name,double base) throws WrongStockBasePriceException {
          if (base < 0)
            throw new WrongStockBasePriceException(base);
          base_ = base;              ref_ = 0;
          price_ = base;             name_ = name;
        }
        public synchronized double getPrice() { return price_; }
        public synchronized void incRef() { ref_++;  resetPrice(); }
        public synchronized void decRef() {
          if (ref_ > 0) {
            ref_--;
            resetPrice();
          }
        }

        private synchronized void resetPrice() {
          price_ = base_ + 0.1 * ref_;
```

```
    }

    public boolean equals(Object o) {
      return name_.equals(((Stock)o).name_);
    }
  }
```

המחלקה BankAccount מייצגת חשבון של לקוח בבנק, החשבון מורכב ממספר חשבון (ID_), סכום הכסף בחשבון עו"ש (savings_), וטבלת המניות שבבעלות בעל החשבון (stocks_). הטבלה ממפה מניה, למספר היחידות שנקנו על ידי הלקוח. הבנק מאפשר משיכת יתר (אוברדרפט) שגובהה מוגבל על ידי השדה maxOverDraft_.

```
class BankAccount {
    private final int ID_;
    private double savings_;
    private double maxOverDraft_;
    private HashMap<Stock,Integer> stocks_;
    //@INV: savings_ + maxOverDraft_ > 0

    BankAccount(int ID, double amount, double maxOverDraft)   {
      if (ID  < 1 || maxOverDraft < 0 || amount < maxOverDraft )
        throw new WrongAcountDataException();
      ID_ = ID;     savings_ = amount;   maxOverDraft_ = maxOverDraft;
      stocks_ =  new HashMap<Stock,Integer>();
    }
    public synchronized void incSaving(double amount) { savings_ += amount; }
    public synchronized void decSaving(double amount) { savings_ -= amount; }
    public synchronized void buyStock(Stock stock) {
        if ((savings_ + maxOverDraft_)  >= stock.getPrice()) {
            Integer num = stocks_.get(stock);
            if (num == null)
              stocks_.put(stock,new Integer(1));
            else
              stocks_.put(stock,new Integer(num.intValue() +1));
            decSaving(stock.getPrice());
            stock.incRef();
        }
    }

    public synchronized void sellStock(Stock stock) {
        Integer num = stocks_.get(stock);
```

```
            if (num != null) {
                if (num.intValue() > 1)
                  stocks_.put(stock, num.intValue() -1);
                else
                  stocks_.remove(stock);
                incSaving(stock.getPrice());
                stock.decRef();
            }
        }
}
```

א. כתבו מתודות testInv() עבור המחלקות BankAccount ו Stock הבודקות את קיום האינווריאנטה של המחלקות מעל שדותיהן (6 נקודות)

המחלקה Dealer מייצגת אובייקט אקטיבי המכיל חשבונות בנק ומניות, ותור של הוראות לביצוע (אובייקטים המממשים את הממשק Command), ומבצע את ההוראות על חשבונות הבנק והמניות, על ידי הפעלת מתודת apply של Command.
המחלקה SynchRandomCollection הינה מבנה נתונים המממש באופן מסונכרן ותחת מנגנון wait/notify פעולות הוספה והסרה של איברים מאוסף, כאשר פעולת add(Object) מוסיפה איבר לאוסף, ופעולת remove() מסירה איבר אקראי כלשהוא מהאוסף.
כמו כן המחלקהSynchMap הינה מימוש של Map כך שכל הפעולות עליה, ובפרט המתודה get(Object), מסונכרות.

```
interface Command {
  public void apply(SynchMap<Integer,BankAccount> accounts,
                    SynchMap<String,Stock> stocks);
}

class Dealer extends Thread {

      SynchMap<String,Stock> stocks_;
      SynchMap<Integer,BankAccount> accounts_;
      SynchRandomCollection <Command> commands_;

      Dealer(SynchMap<Integer,BankAccount> accounts,
             SynchMap<String,Stock> stocks,
             SynchRandomCollection<Command> commands)
        { stocks_ = stocks;  accounts_ = accounts;  commands_ = commands; }

      public void run() {
        while (true) {
          Command command  = commands_. remove();
          command.apply(accounts_,stocks_);
```

```
        }
      }
    }
```

נתון תהליך בו רצים שני Dealers מעל אותה קבוצת חשבונות, אותה קבוצת מניות, ואותה קבוצת פקודות.
כמו כן נתון כי קיימים רק המימושים הבאים ל Command:

```
class BuyCommand implements Command  {
    protected Integer accountID_;
    protected String stockName_;
    BuyCommand(int accountID,String stockName) {
            accountID_ = new Integer(accountID);
            stockName_ = stockName;
    }

    public void apply(SynchMap<Integer,BankAccount> accounts ,
                    SynchMap<String,Stock> stocks)    {
            BankAccount account = accounts.get(accountID_);
            Stock stock = stocks.get(stockName_);
            if (account != null &&  stock  != null)
              account.buyStock(stock);
    }
}
```

```
class SellCommand  implements Command {
    protected Integer accountID_;
    protected String stockName_;
    SellCommand(int accountID,String stockName)  {
            accountID_ = new Integer(accountID);
            stockName_ = stockName;
    }
    public void apply(SynchMap<Integer,BankAccount> accounts ,
                    SynchMap<String,Stock> stocks)    {
            BankAccount account = accounts.get(accountID_);
            Stock stock = stocks.get(stockName_);
            if (account != null &&  stock  != null)
              account.sellStock(stock);
    }
}
```

ב. האם המחלקות Stock ו BankAcount בטוחות מבחינת שמירה על האינווריאנטה, בהרצת התהליך הנ"ל? (8 נקודות)
ג. האם הרצת התהליך הנ"ל (על קבוצת פקודות נתונה) נכונה מבחינת התאמת כל ביצוע מקבילי אפשרי לביצוע סדרתי, על סדר פעולות כלשהוא של קבוצת הפקודות. אם כן הסבירו למה, אם לא הביאו דוגמא נגדית (8 נקודות)
ד. אם קיימת בעיה בסעיפים ב ו/או ג, פתרו אותה. (8 נקודות)

## שאלה 2 (30 נקודות)

בשאלה זו אנו מדמים את פעולתה של מעלית (כמו זו המשרתת אתכם בעלותכם למעבדות בבנין 34).
מעלית (Elevator) מוגדרת על ידי מספר הקומות בבניין, הקומה הנוכחית בה היא נמצאת, והמצב בו היא נמצאת: עולה(GOINGUP), יורדת(GOINGDOWN), או עוצרת(IDLE).
נוסע בתוך המעלית יכול לבקש עצירה בקומה מסוימת (= לחיצה על כפתור המציין קומה), בעוד שנוסע מחוץ למעלית יכול לבקש שהמעלית תעצור בקומה בה הוא נמצא לצורך עליה או ירידה (= לחיצה על כפתור ↑ או ↓).
בתכנון המעלית הושם דגש על יעילות הפעלתה, כך שהמעברים למעלה ולמטה יצומצמו עד למינימום. לשם כך, מכיל המצב הפנימי של המחלקה Elevator אובייקט מטיפוס Trajectory המייצג מסלול של תחנות לעצירה. למסלול יש כיוון (UP,DOWN) וקבוצה של קומות בהן יש לעצור במסלול זה. בכל נסיעה של המעלית למעלה או למטה, מיוצר אובייקט 'מסלול' שכזה עבור כיוון המעלית באותו זמן, עם תחנות העצירה.
בקשות לעצירה המתקבלות למעלית תוך כדי נסיעה, מוספות למסלול הנסיעה אם הן בכיוון הנסיעה, אחרת הן נשמרות ברשימה pendingRequests_ לשם העברתם בהמשך למסלול נסיעה אחר.
קיימים שני סוגים של בקשות, המתאפיינים על ידי השדה (type_) במחלקה Request:

- GO - קריאה מתוך המעלית לנסיעה לקומה היעד (floor_)
- CALL - בקשה מחוץ למעלית, לעצירה בקומה בה נמצא המבקש (floor_), לשם ירידה או עליה (ערכי UP/DOWN של השדה direction_)

```
#include <set>
#include <vector>
#include <iostream>

enum Direction { UP, DOWN, NONE };
enum RequestType { GO, CALL };

class Request;
typedef std::vector<const Request*> Requests;



class Trajectory
{
public:
  typedef std::set<int> Stops;
  Trajectory(Direction direction) : direction_(direction) {}
  Direction getDirection() const { return direction_; }
```

```
  // While in currentFloor, add a new stop request on the trajectory
  // The request is ignored if it cannot be added to the trajectory
  // Return true if the trajectory is modified
  bool addStop(int currentFloor, int floor) {
    if (direction_ == DOWN) {
            if (floor < currentFloor) {
              stops_.insert(floor);
              return true;
            }
    } else  if (direction_ == UP) {
          if (floor > currentFloor) {
                    stops_.insert(floor);
                    return true;
          }
    }
    return false;
  }

  // Is the trajectory empty?
  bool empty() const {  return stops_.empty(); }

  // What is the next stop to which we want to stop?
  int nextStop(int currentFloor) const {
    if (direction_ == UP)
      return (*stops_.lower_bound(currentFloor));
    else if (direction_ == DOWN)
      return (*stops_.upper_bound(currentFloor));
    else
       return -1;
  }

  // Remove element to which i points from the set
  void removeStop(int floor) {
    stops_.erase(floor);
  }

private:
  const Direction direction_;
Stops stops_; //Sorted list of floors on which the trajectory will stop.
};
```

```
-----------------------------------------------------------------------------------------------------------------------------------------

// A request sent to the elevator:
// - Goto a certain floor
// - Call the elevator from a certain floor to go up or down
class Request {
public:
  Request(RequestType rt, int floor) : type_(rt), floor_(floor), direction_(NONE) {}
  Request(RequestType rt, int floor, Direction direction) : type_(rt), floor_(floor), direction_(direction) {}
  int getFloor() const {return floor_;}
  Direction getDirection() const {return direction_;}
  RequestType getType() const {return type_;}
private:
  const int floor_;
  const Direction direction_;
  const RequestType type_;
};

-----------------------------------------------------------------------------------------------------------------------------------------

class Elevator {
public:
  enum State {GOINGUP,GOINGDOWN,IDLE};
  Elevator(int numFloors) : numFloors_(numFloors), currentFloor_(0),state_(IDLE),currentTraj_(UP) {}

  // Remember that a request has been sent to the elevator
  // If the request is compatible with the current trajectory,
  // add it for immediate handling
  // else remember it for one of the next trajectories
  void addRequest(const Request& r) {
    if (canAddRequestToTraj(r)) {
      currentTraj_.addStop(currentFloor_, r.getFloor());
    } else {
      pendingRequests_.push_back(&r);
    }
  }

  // Perform the requests: return after an adjacent floor has been
  // reached or if there are no requests to service.
```

```
  void handleRequests(Requests* newRequests);

private:
  const int numFloors_;
  int currentFloor_;
  State state_;
  Trajectory currentTraj_;
  Requests pendingRequests_;

  // Can we reach target from current by going in direction direction?
  bool follow(int targetFloor, int currentFloor, Direction direction) {
    if (direction == UP)
      return (targetFloor > currentFloor);
    else
      return (targetFloor < currentFloor);
  }

  // Can a request be added to the current trajectory?
  bool canAddRequestToTraj(const Request& r) {
    if ((r.getType()==CALL && r.getDirection()==currentTraj_.getDirection()) || r.getType() == GO)
      return follow(r.getFloor(),currentFloor_,currentTraj_.getDirection());
    else
       return false;
  }
};
```

א. כדי להבטיח פעולה תקינה של המעלית, נדרשים unit tests, עבור המחלקה Trajectory הבודקים את תנאי הסיום של המתודות. על ה unit tests לכסות את כל התרחישים האפשריים בכל מתודה. עבור המתודה ()empty לדוגמא, הוגדרו שני מבחנים, הבודקים את שני התרחישים האפשריים עבורה: תור ריק אם לא הוכנסה שום תחנת עצירה, ותור שאינו ריק אם הוכנסה תחנה שכזו.

```
void testEmptyTraj1() {
    Trajectory t1(UP);
    t1.addStop(1, 3);
    if (t1.empty()) {
        std::cerr << "Error: empty trajectory after addStop" << std::endl;
    } else {
        std::cout << "EmptyTraj1 passed" << std::endl;
    }
}
```

```
void testEmptyTraj2() {
    Trajectory t1(UP);
    if (! t1.empty()) {
        std::cerr << "Error: trajectory not empty when constructed" << std::endl;
    } else {
        std::cout << "EmptyTraj2 passed" << std::endl;
    }
}
```

כתבו unit tests עבור שאר המתודות: addStop(), removeStop(), nextStop(). (12 נקודות)

ב. התבוננו בקטע הקוד הבא, המייצר מעלית ומפעיל אותה על פי תרחיש נתון. בתוכנית מוגדר שעון המדמה את גורם הזמן. בכל יחידת זמן עולה המעלית או יורדת קומה, או נשארת בקומתה הנוכחית, כך שבסוף כל פעימת שעון נמצאת המעלית בקומה כלשהיא.

המתודה getRequests(int clock) מחזירה רשימה של בקשות שנעשו במהלך יחידת הזמן [clock-1,clock). המתודה handleRequests(Requests* newRequests) במחלקה Elevator, מקבלת רשימה זו, ופועלת בתבנית הבאה:

- העברת הבקשות החדשות המתאימות למסלול הנוכחי (currentTraj_) או למסלול חדש המיוצר עקב סיום המסלול הקודם, או שימורן ברשימת ההמתנה (pendingRequests_).
- עדכון מצב המעלית, בהתאם למצבה הקודם, ולתוכן המסלול הנוכחי ו/או רשימת ההמתנה.
- אם המעלית עוברת למצב עצירה (IDLE) ממצב עליה (GOINGUP) או ממצב ירידה (GOINGDOWN) יש להדפיס הודעה על כך.

<u>הערה</u>: כמוזכר לעיל, עצירה של המעלית בקומה נמשכת לפחות זמן שעון אחד.

ממשו את המתודה handleRequests במחלקה Elevator, יש לדאוג לכך שכל הזיכרון המוקצה בהרצת ה main() ישתחרר (18 נקודות)

```
// Get random requests arriving between clock-1 and clock
Requests* getRequests(int clock) {
  Requests* v = new Requests();

  switch (clock)
  {
    case 1:
      v->push_back(new Request(CALL,1,UP));
      v->push_back(new Request(CALL,3,DOWN));
      break;
    case 2:
      v->push_back(new Request(GO,2));
      v->push_back(new Request(CALL,2,DOWN));
      break;
```

```
      case 3:
        v->push_back(new Request(GO,2));
        v->push_back(new Request(CALL,2,DOWN));
        break;
   }

   return v;
}

int main() {
   Elevator elev(3);   // Elevator moves between floors 0 and 3 included
   int clock = 0;   // The clock of the simulation.

   // Main simulation loop
   while (true) {
      clock++;      // New round
      // Get all requests until time clock, and handle the requests
      elev.handleRequests(getRequests(clock));
   }
   return 0;
}
```

## שאלה 3 (30 נקודות)

התבוננו בשתי התוכנית הבאות הממשות client לתהליך של server כלשהוא:

```
class Client1  {
 public static void main(String args[]) {
   try {
      Socket socket = new Socket("tapuz",1300);
      {
          //do something
      }
      socket.close();
   } catch (Exception e) {
        e.printStackTrace();
   }
 }
}
```

```
class Client2  {
 public static void main(String args[]) {
   try {
      DatagramSocket socket = new DatagramSocket();
      {
        // do  something
      }
      socket.close();
   } catch (Exception e) {
        e.printStackTrace();
   }
 }
}
```

א. איזה סוג קשר (connection) יוצרת כל אחת מהתוכניות עם השרת? (3 נקודות)
ב. הגדירו את רמת האמינות (reliability) של החלפת הודעות בין הלקוח לשרת בכל אחד משני הקשרים בסעיף א, ופרטו כיצד משיג הפרוטוקול ב transport layer רמת אמינות זו עבור כל אחד מהם. (12 נקודות)
ג. עבור כל תוכנית, ציינו האם, באחת משורות הקוד המופיעות בה במפורש, היא עלולה להיקלע ל I/O blocking, תוך הצבעה על שורת הקוד המתאימה ועל הסיבה לכך (5 נקודות)
ד. אם תוכנית עלולה להיקלע לI/O blocking, שנו אותה כך שימנע I\O blocking זה (10 נקודות)

חומר עזר:

1. מתודות ושדות נבחרים של המחלקה SelectionKey

| | |
|---|---|
| static int | **OP_ACCEPT**<br>Operation-set bit for socket-accept operations. |
| static int | **OP_CONNECT**<br>Operation-set bit for socket-connect operations. |
| static int | **OP_READ**<br>Operation-set bit for read operations. |
| static int | **OP_WRITE**<br>Operation-set bit for write operations |

| | |
|---|---|
| boolean | **isAcceptable**()<br>Tests whether this key's channel is ready to accept a new socket connection. |
| boolean | **isConnectable**()<br>Tests whether this key's channel is ready, so that invoking finishConnect() does not block. |

| | |
|---|---|
| boolean | **isReadable**()<br>Tests whether this key's channel is ready for reading. |
| boolean | **isWritable**()<br>Tests whether this key's channel is ready for writing. |

2. מתודות נבחרות של המחלקה SocketChannel

| | |
|---|---|
| static `SocketChannel` | **open**()<br>Opens a socket channel. |
| `SelectorProvider` | **configureBlocking**(boolean block)<br>Adjusts this channel's blocking mode. |
| `SelectionKey` | **register**(`Selector` sel, int ops, `Object` att)<br>Registers this channel with the given selector, returning a selection key. |
| abstract void | **cancel**()<br>Requests that the registration of this key's channel with its selector be cancelled. |
| abstract boolean | **connect**(InetSocketAddress remote)<br>Connects this channel's socket. Return `true` if a connection was established, `false` if this channel is in non-blocking mode and the connection operation is in progress |
| abstract boolean | **finishConnect**()<br>Finishes the process of connecting a socket channel. `Return true` if, and only if, this channel's socket is now connected |

## שאלה 4 (10 נקודות)

בחברת תרופות גדולה (שמה המדויק יינתן, לפי דרישה, לאחר תקופת המבחנים), הוחלט לאחסן את נתוני התרופות בבסיס נתונים, לשם מעקב על תרופות שפג תוקפן.
תרופה מוגדרת על ידי שם, ונוסחת התרכובת של התרופה.
בכל פעם שמופעל קו הייצור של תרופה, ניתן מספר קטלוגי לקבוצת התרופות שיוצרה בפעם זו, תוך ציון מספר העותקים שיוצרו, ותאריך הייצור ותאריך התפוגה של קבוצה זו.

א. הגדירו מודל נתונים לשמירת המידע הנ"ל באופן שלם ומינימאלי. ההגדרה תתבסס על טבלאות, שדות, מפתחות ראשיים וזרים - כפי שנלמד בכיתה. (5 נקודות)

ב. כתבו שאילתת SQL המציגה עבור כל תרופה (יש לציין את שמה), את המספר הקטלוגי של הקבוצה שתוקפה פג ביום הבחינה (אם יש כזו), ואת הכמות של התרופות שיוצרו בכל קבוצה שכזו. על הרשימה להיות ממוינת על פי שם התרופה בסדר עולה, ועל פי כמות הייצור בסדר יורד (5 נקודות)