אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

רשמו תשובותיכם בגיליון התשובות בלבד
תשובות מחוץ לגיליון לא יבדקו.
שימו לב:
על תשובות ריקות יינתן 20% מהניקוד!

בהצלחה!

תאריך הבחינה: 24.2.2014
שם המורה: ד"ר אנדרי שרף
ד"ר דוד טולפין
ד"ר מני אדלר
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשע"ד
סמסטר: א'
מועד: ב'
משך הבחינה: שלש שעות
חומר עזר: אסור

שאלה 1 (30 נקודות)

בשאלה זו נמשיך לעסוק בסימולציית מעבר הצומת של מועד א.

כזכור מכונית מאופיינת על ידי מיקום, ויכולת התקדמות:

```
interface Car {
  Position getPosition(); // returns the position of the car
  // increase the position of the car by one. if impossible, throws CarAdvancedException
  void advance() throws CarAdvancedException;
}

interface Position {
        int getX(); // gets the horizontal position
        int getY(); // gets the vertical position
        void setX(int x); // sets the horizontal position
        void setY(int y); // sets the horizontal position
}
```

המחלקה TrafficLight מסמלצת רמזור, המחליף את צבעיו על פי מרווחי זמן בקבועים מראש.

```
enum Light {RED, YELLOW, GREEN}
```

```
class TrafficLight implements Runnable {
        public static TrafficLight create(int redInterval, int yellowInterval, int greenInterval) {
                TrafficLight tl = new TrafficLight(redInterval,yellowInterval,greenInterval);
                new Thread(tl).start();
                return tl;
```

```
        }
        private TrafficLight(int redInterval, int yellowInterval, int greenInterval) {
                _light = Light.RED;
                _redInterval = redInterval; _yellowInterval = yellowInterval; _greenInterval = greenInterval;
        }

        public Light getLight() { return _light; }

        public void run() {
                while (true){
                 try {
                        Thread.sleep(_redInterval);
                        _light = Light.YELLOW;
                        Thread.sleep(_yellowInterval);
                        _light = Light.GREEN;
                        Thread.sleep(_greenInterval);
                        _light = Light.RED;
                 } catch (InterruptedException e) {
                        return;
                 }
                }
        }
        Light _light;
        final int _redInterval, _yellowInterval, _greenInterval;
}
```

הממשק Junction מגדיר צומת. בצומת ארבעה רמזורים, אחד לכל כיוון.
ניתן להניח כי הצומת ממוקמת תמיד ב (0,0) . הצומת והמכוניות אינם תופסים מקום. לדוגמא, מכונית מכיוון LEFT עוברת צומת מ: 1,0- ל: 1,0.
מכונית המגיעה מכיוון מסוים ומעוניינת לחצות את הצומת מצטרפת לרשימת ההמתנה עבור כיוון זה ע"י הפעלת המתודה addCar.
המתודה move מעבירה, כאשר האור מתחלף לירוק, מכוניות מרשימת ההמתנה של הכיוון הנתון אל מעבר לצומת, ע"י הפעלת מתודת ה advanced שלהם. מכונית יכולה רק להתקדם בכיוון שלה (=אין בצומת פניות ימינה/שמאלה).
המתודה מסתיימת כאשר צבע הרמזור מתחלף לאדום.
מכוניות בכיוונים מקבילים (ימין ושמאל, למעלה ולמטה, וכו') יכולות לנסוע באותו זמן, אך לא בכיוונים חותכים (ימין ולמעלה, שמאלה ולמטה וכו').

```
enum Direction {LEFT, RIGHT, UP, DOWN}
interface Junction {
 void add(Car car, Direction dir) throws AddException; // add a car to the waiting list of the given direction
 void move(Direction dir) throws MoveException; // move waiting cars from the given direction, as long as possible
}
```

במועד א הגדרנו את תנאי ההתחלה והסיום של הממשק (אשר התבססו על שאילות נוספות שהוספו לממשק):

```
for each car in getCars(Direction.LEFT)
   car.getPosition().equals(-1,0) &&
for each car in getCars(Direction.RIGHT)
   car.getPosition().equals(1,0) &&


Note, that in this invariant version, there's no restriction on the traffic lights

@PRE:
car != null
((dir == Direction.UP &&   car.getPosition().equals(0,1)) ||
(dir == Direction.DOWN &&   car.getPosition().equals(0,-1)) ||
(dir == Direction.LEFT &&   car.getPosition().equals(-1,0)) ||
(dir == Direction.RIGHT &&   car.getPosition().equals(1,0))) &&
!getCars(dir).contains(car)

@POST:
getCars(dir).contains(car) && car.getPosition().equals(@PRE(car.getPosition()))
void add(Car car, Direction dir) throws AddException;


@PRE:
getLight(dir) = Light.GREEN &&
for each dir  in oppositeDirs(dir)
   light == Light.RED || light == Light.YELLOW) &&
!getCars(dir).isEmpty()

@POST:
for each car in (@PRE(getCars(dir) - getCars(dir))
   @PRE(car).advanced() → car.getPosition().equals(@PRE(car).getPosition())
void move (Direction dir) throws MoveException;
```

להלן מימוש המחלקה SimpleJunction המממשת את הממשק Junction.

אם זה עוזר, אז מדובר בקוד שניתן בפתרון המבחן, תוך השמטת השימוש ב OrientationSemaphore

```
public class SimpleJunction implements Junction {

public SimpleJunction(TrafficLight leftLight, TrafficLight rightLight,
            TrafficLight upLight, TrafficLight downLight) {
            _cars = new HashMap<Direction,List<Car>>();
            _cars.put(Direction.LEFT, new LinkedList<Car>());
            _cars.put(Direction.RIGHT, new LinkedList<Car>());
```

```
		_cars.put(Direction.UP, new LinkedList<Car>());
		_cars.put(Direction.DOWN, new LinkedList<Car>());
		_lights = new HashMap<Direction,TrafficLight>();
		_lights.put(Direction.LEFT, leftLight);
		_lights.put(Direction.RIGHT, rightLight);
		_lights.put(Direction.UP, upLight);
		_lights.put(Direction.DOWN, downLight);
	}

	public void add(Car car, Direction dir) {
		List<Car> cars = _cars.get(dir);
		synchronized (cars) {
			cars.add(car);
			cars.notifyAll();
		}
	}

	public void move(Direction dir) throws MoveException { //advance cars from the given direction it possible
		// 1. wait for a green light
		TrafficLight light = _lights.get(dir);
		synchronized (light) {
			try{
				while (light.getLight() != Light.GREEN)
					light.wait();
			} catch (InterruptedException e) {
				return;
			}
		}

		//2. pass cars, as long as the light is green, if possible
		List<Car> cars = _cars.get(dir);
		while (true) {
			synchronized (cars) {
				Car car = null;
				//2.1  wait for cars
				try{
					while (cars.isEmpty())
						cars.wait();
					car = cars.remove(0);
				} catch (InterruptedException e) {
					return;
				}
				//2.2 advance one car
				if (light.getLight() == Light.GREEN)
```

```
                                synchronized (car) { car.advance(); }
                        else {
                                // The TrafficLight interrupts all move threads when the light turns RED
                                 // In case the light turned to RED before the car was advanced,
                                 // return car back to the waiting list
                                synchronized (cars) { cars.add(0, car); }
                                return;
                        }
                }
        }
}

Map<Direction,List<Car>> _cars;
Map<Direction,TrafficLight> _lights;
}
```

א. האם המחלקה בטוחה תחת הרצה מקבילית? נמקו בקצרה [5 נקודות]

ב. הרחיבו את האינווריאנטה של הממשק Junction, כך שלא יתאפשר מצב בו לרמזורים עם אוריאנטציה שונה (למעלה-שמאלה, למטה-ימינה וכו') יהיה במקביל אור ירוק. [5 נקודות]

ג. האם הקוד הנוכחי שומר על בטיחות, לאור התנאי החדש? נמקו בקצרה [5 נקודות]

ד. עדכנו את הקוד, כך שתישמר הבטיחות [15 נקודות]

שאלה 2 (30 נקודות)

לאחר שחברת שחררוני פשטה את הרגל הוחלט על מספר צעדי התייעלות והבראה. בראשם הוחלט להוסיף מחלקת אב Statistics אשר תעקוב אחרי המשחק, ולממש את מבנה הנתונים **rowscols_** בעצמם (ללא שימוש במחלקה vector). להלן הממשק החדש:

```
class Statistics{
public:
        Statistics(){;}
        virtual ~Statistics(){;}
        virtual void computeStatistics()=0;
        void setRate(float rate){_rate=rate;}
        void setDistribution(float dist){_distribution=dist;}
private:
        float _rate;
        float _distribution;
};
class Cell{
public:
        Cell(){;}
        virtual ~Cell(){;}
        void SetX(int x){_x=x;}
        void SetY(int y){_y=y;}
        int GetX(){return _x;}
        int GetY(){return _y;}
        virtual void SetCell()=0;
        virtual void UnsetCell()=0;
private:
        int _x;
        int _y;
};
class BlockCell: public Cell, public Statistics{
public:
        BlockCell(){;}
        BlockCell(Cell *cell){;}
        ~BlockCell(){;}
        virtual void SetCell() {_blocked=1;}
        virtual void UnsetCell(){_blocked=0;}
        virtual void computeStatistics(){setRate(0);setDistribution(0);}
private:
        bool _blocked;
};
class  BlockScene{
public:
```

```
        BlockScene(int nrows, int ncols);
        BlockScene(const BlockScene &other);
        ~BlockScene();
        BlockScene& operator=(BlockScene &other);
private:
        //a 2D grid of size nrows X ncols consisting of blockcells
        Cell***         _rowscols;
        int             _nrows;
        int             _ncols;
};
```

א. עזרו למפתחים להשלים עבור המחלקה BlockScene וממשו 1. בנאי (constructor), 2. בנאי מעתיק (copy-constructor), 3. אופרטור= 4. הורס (destructor). [12 נקודות]
הנחיות:
- יש לאתחל את התאים כרצונכם.
- כאשר נדרש יש להשתמש בהעתקה עמוקה.

ב. כיצד נראית תמונת הזכירון עבור מופע בודד של BlockCell (בשורה @@ קוד). הראו בטבלאות את סידור המשתנים והפונקציות בזיכרון כפי שיתקבל. [12 נקודות]

```
void main(){
        BlockCell cell1;
        //@@
}
```

ג. סמנו מה התשובה הנכונה לגבי קטע הקוד הבא (בהנחה שהמימוש תקין): [6 נקודות]

```
BlockScene& test_dummy(BlockCell cell){
        BlockScene scn2;
        return scn2;
}
void main(){
        BlockCell cell1;
        BlockScene scn1 = test_dummy(cell1);
}
```

1. הקוד לא יתקמפל מכיוון שcell1 נשלח by val ללא הגדרת בנאי מעתיק
2. הקוד לא יתקמפל מכיוון שאסור לרשת משני אבות בC++
3. ישנה בעיית זיכרון על הSTACK אך לא ניתן לומר בוודאות מה יקרה בזמן ריצה
4. ישנה דליפת זיכרון בעת העברת cell1 בתור פרמטר לפונקציה
5. כל התשובות נכונות

שאלה 3	(30 נקודות)

בתרגיל האחרון עסקנו בפרוטוקול ה STOMP התומך בפרסום של הודעה לתור של מנויים.
בשאלה זו, נעסוק בגרסה פשוטה יותר של פרוטוקול זה.

התחברות לשרת נעשית על ידי שליחת ההודעה COONECT לשרת ה STOMP

```
CONNECT
login: <username>
passcode:<passcode>

^@
```

לשם פשטות השאלה, בגירסה זו של הפרוטוקול, השרת <u>אינו</u> מחזיר הודעה כפי שמימשתם בתרגיל.

הצטרפות כמנוי לתור נעשית על ידי שליחת ההודעה SUBSCRIBE לשרת ה STOMP:

```
SUBSCRIBE
destination: <queue name>

^@
```

ביטול רישום לתור ניתן על ידי שליחת ההודעה UNSUBSCRIBE לשרת ה STOMP:

```
UNSUBSCRIBE
destination: <queue name>

^@
```

שליחת הודעה לקבוצת מנויים רשומים, ניתנת על ידי שליחת הודעת SEND ל שרת ה STOMP המכילה את שם קבוצת המנויים, ואת תוכן ההודעה למשלוח.

```
SEND
destination: <queue name>

<message string>
^@
```

כתגובה, ישלח השרת הודעת MESSAGE לכל אחד מהלקוחות שנרשמו לתור

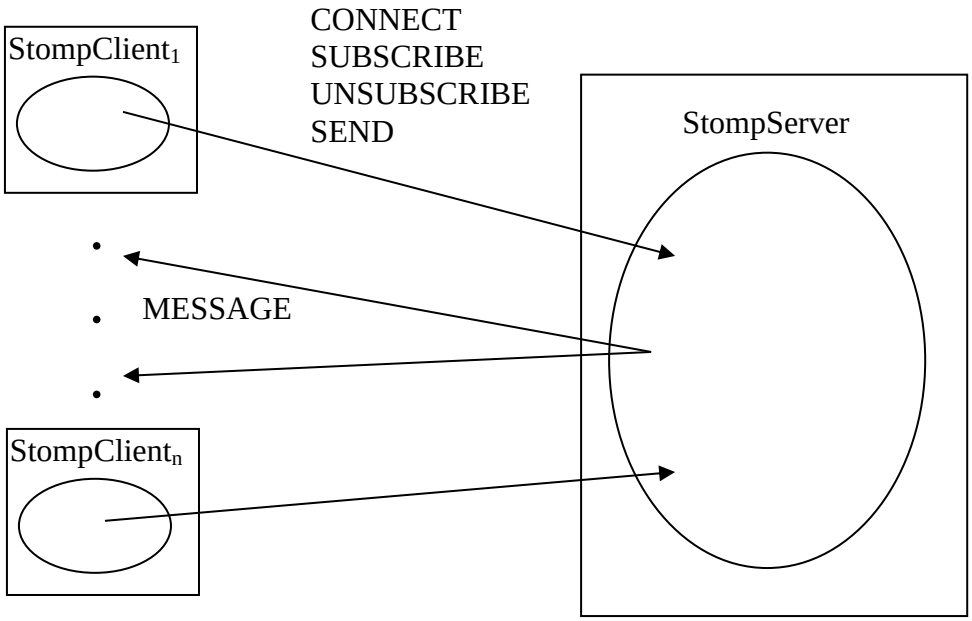
```
MESSAGE
destination: <queue name>
message-id: <message-identifier>

<message string>
^@
```

על פי הפרוטוקול, על כל אחת מבקשות הלקוח (CONNECT, SUBSCRIBE, UNSUNSCRIBE, SEND) עשוי השרת להחזיר הודעת שגיאה (ERROR).
בשאלה זו נניח כי הבקשות תמיד מצליחות, כך שלא נשלחות אף פעם הודעות שגיאה.
ההודעות היחידות שנשלחות מהשרת הינן הודעות מסוג MESSAGE.

באופן זה יכולים לקוחות שונים להירשם לתורים שונים, לשלוח הודעות לתורים, ולקבל הודעות שנשלחו לתורים אליהם הן נרשמו.

נניח כי קיים מימוש של שרת STOMP המקבל ושולח הודעות ב TCP – כמו זה שמימשתם בתרגיל.



TCP Connection

במודל זה, ה StompClient מגדיר TCP Socket דרכו הוא שולח ומקבל הודעות, בפורמט של STOMP אשר תואר לעיל.

כדי לפשט את הגישה לשרת הוחלט להגדיר תהליך ביניים בשם StompClientRMI, המכיל Remote Object (RMI) המממש ממשק התחברות לשרת ה Stomp

```
public interface StompConnector extends java.rmi.Remote {
   StompOperator connect(String login, String passcode)  throws java.rmi.RemoteException,IOException;
}
```

המתודה connect מחברת את הלקוח לשרת ה Stomp על ידי שליחת ההודעה CONNECT (כזכור, אני מניחים כי הפעולה תמיד מצליחה. וכן אין הודעת CONNECTED מהשרת).
המתודה מחזירה RemoteObject המממש ממשק לפעולות הלקוח מול שרת ה Stomp, על פי הפרוטוקול:

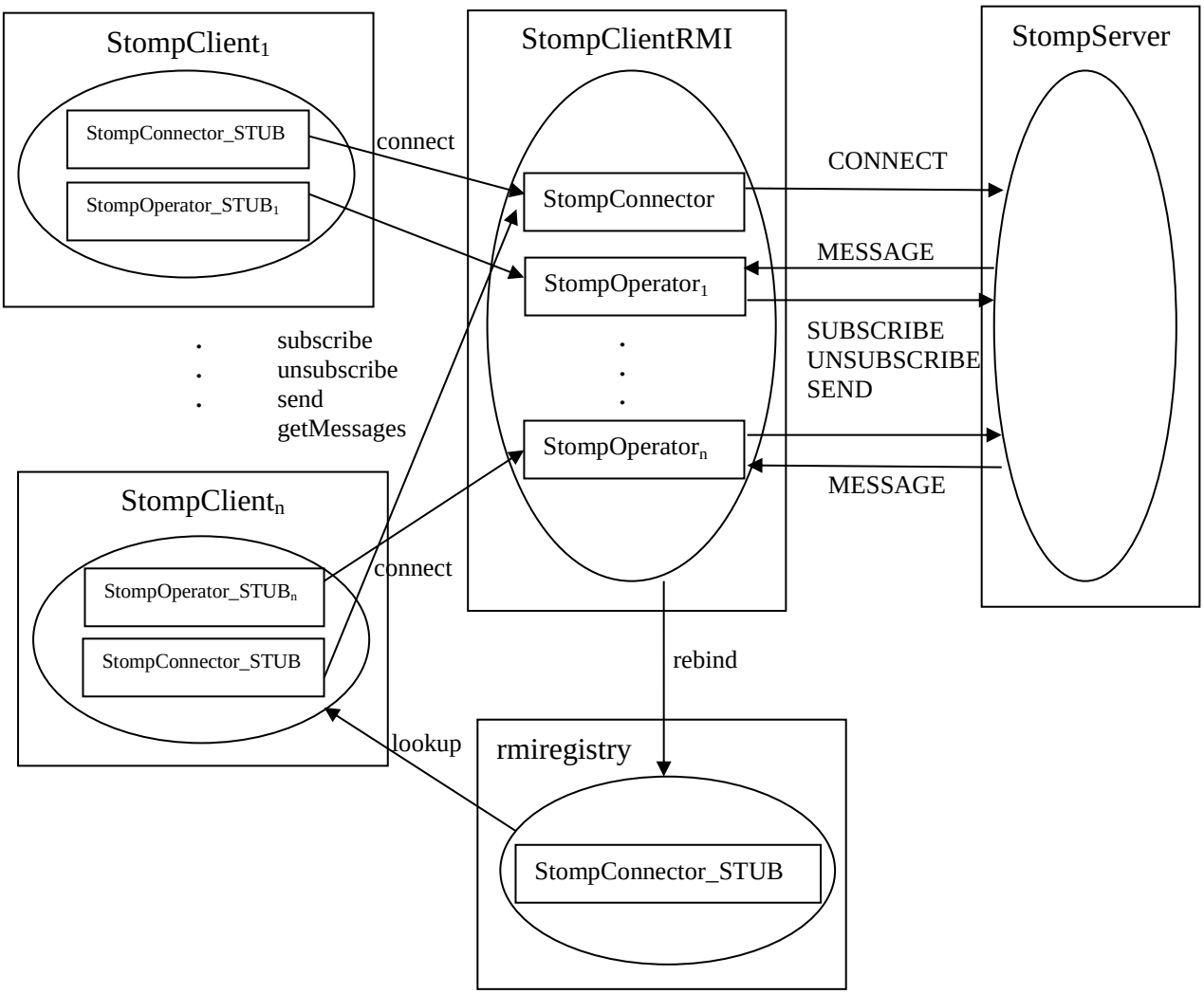
```
public interface StompOperator extends java.rmi.Remote {
   void subscribe(String groupName) throws java.rmi.RemoteException,IOException;
   void unsubscribe(String groupName) throws java.rmi.RemoteException,IOException;
   void send(String groupName, String str) throws java.rmi.RemoteException,IOException;
```

```
  List<String> getMessages() throws java.rmi.RemoteException,IOException;
}
```

המתודות **subscribe, unsubscribe, send** שולחות לשרת את ההודעות SUBSCRIBE, UNSUBSCRIBE, SEND. המתודה **getMessages** מחזירה את תוכן ההודעות שהגיעו מהשרת (ע"י פעולת MESSAGE) ועדיין לא נקראו על ידי הלקוח.

ה **StompClientRMI** מתווך בין בין ה **StompClient** ל **StompServer**, באופן הבא:

- הוא מגדיר אובייקט הממש את **StompConnector** ומפרסמו ב **rmiregistry**.
- ה **StompClient** ניגש ל **rmiregistry** ומקבל את ה **StompConnector** כך שהוא יכול לבצע את פעולת ההתחברות, ולקבל כערך מוחזר אובייקט הממש את **StompOperator** לשם ביצוע פעולות באופן נוח מול שרת ה Stomp.



RMI Connection **TCP Connection**

להלן מימוש הממשק **StompConnector**

```
public class StompConnectorImpl extends java.rmi.server.UnicastRemoteObject implements StompConnector {
  protected String _stompServerHost;
  protected int _stompServerPort;

  public StompConnectorImpl(String stompServerHost, int stompServerPort)  throws java.rmi.RemoteException {
    _stompServerHost = stompServerHost;
    _stompServerPort = stompServerPort;
  }

  public StompOperator connect (String login, String passcode) throws java.rmi.RemoteException,IOException {
    Socket socket = new Socket(_stompServerHost, _stompServerPort);
    PrintWriter out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream(),"UTF-8"));
    out.print("CONNECT\nlogin: " + login + "\npasscode: " + passcode + "\n\n" + '\0');
    return new StompOperatorImpl(socket.getInputStream(),socket.getOutputStream());
  }
}
```

א. המחלקה StompOperatorImpl מממשת את הממשק StompOperator . השלימו את מימוש המתודות subscribe, unsubscribe, send (אין צורך לממש את getMessages) [6 נקודות]

```
public class StompOperatorImpl
  extends java.rmi.server.UnicastRemoteObject
  implements StompOperator
{
  protected PrintWriter _writer;
  protected BufferedReader _reader;

  public StompOperatorImpl(InputStream in, OutputStream out) throws java.rmi.RemoteException, IOException {
    _reader = new BufferedReader(new InputStreamReader(in,"UTF-8"));
    _writer = new PrintWriter(new OutputStreamWriter(out,"UTF-8"));
  }

  public void subscribe (String group)   throws java.rmi.RemoteException, IOException {
      //@TODO
  }

  public void unsubscribe (String group)  throws java.rmi.RemoteException, IOException {
     //@TODO
  }

  public void send (String group, String str)   throws java.rmi.RemoteException, IOException {
      //@TODO
```

```
  }

  public List<String> getMessages() throws java.rmi.RemoteException {
        // never mind
  }
}
```

ב. השלימו את התוכנית StompClientRMI. [4 נקודות]
הניחו כי תהליך של rmiregistry רץ ב host שכתובתו "132.23.5.8", על port 2010

```
public class StompClientRMI {
  public static void main(String[] args) {
    try {
      //@TODO
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

ג. השלימו את מתודת ה main של התוכנית StompClient:
- קבלת StompConnector
- התחברות לשרת ה Stomp
- רישום לתורים ששמם q1, q2
- שליחת ההודעה "Suzy Surprise" לתור q3
[4 נקודות]

```
public class StompClient {
  public static void main(String[] args) {
    try {
        //@TODO1: get  StompConnector
        //@TODO2: connect to the stomp server
        //@TODO3: subscribe to queues q1
        //@TODO4: send  "Suzy Se" message to queue q3
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

ד. כמה פעולות תקשורת נדרשות עבור הרצת התוכנית StompClient? פעולת תקשורת מוגדרת כהעברת מידע בפרוטוקול TCP מתהליך לתהליך, בכיוון אחד. [7 נקודות]

ה. ציינו נכון/ לא נכון עבור הקביעות הבאות: [9 נקודות]

I ב StompServer הממומש ע"י reactor יש פחות פעולות תקשורת ממימוש המתבסס על thread-per-client server.

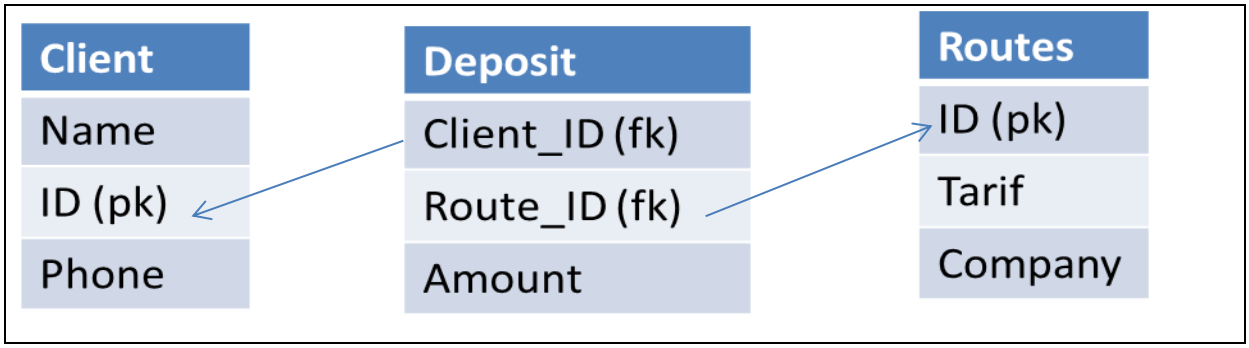
II StompServer הממומש ע"י reactor אמין יותר מכזה המתבסס על thread-per-client server.

III StompServer הממומש ע"י reactor מנצל טוב יותר את המעבד מ StompServer המבוסס על thread-per-client server.

שאלה 4	**(10 נקודות)**

במועד א' הגדרנו מודל נתונים עבור כרטיס נסיעות.
כזכור מודל הנתונים מכיל, עבור כל לקוח, את סכומי הכסף שנותרו עבור כל סוג של נסיעה.

- קיימים סוגים שונים של מסלולי נסיעה, המאופיינים ע"י קוד, תעריף, ושם חברת האוטובוסים המפעילה קו זה.
- לקוח יכול להפקיד סכום כסף לכל סוג של מסלול נסיעה
- לקוח מאופיין ע"י שם, תעודת זהות, וטלפון (למקרה שהכרטיס אבד).



לאחר סיום הפיילוט המוצלח של הכרטיסים החכמים. הוחלט להרחיב את המודל בשני פרמטרים:

- סכום הכסף של הלקוח אינו מוגדר עבור קו ספציפי אחד, אלא עבור קבוצת קווים מאותו סוג (לדוגמא: לכל הקווים העירוניים בתל אביב יש אותו קוד).
- מספר חברות שונות להפעיל את אותו קו (לדוגמא: הנסיעה מבאר שבע לתל אביב, המופעלת ע"י מטרופולין וע"י אגד).

א. הרחיבו את מודל הנתונים כך שיתמוך במודל הנתונים החדש [5 נקודות]

ב. כתבו שאילתת SQL המחזירה את קווי הנסיעה מסוג 17, ואת שמות חברות הנסיעה המפעילות כל קו, ממוין ע"פ תעריפי הנסיעה בסדר יורד [5 נקודות]