גיליון תשובות

מספר נבחן:_____

שאלה 1 (30 נקודות)

א. התכונה הנשמרת מתייחסת אך ורק למיקום המכוניות בצומת, כלומר למיקום המכוניות בשדה cars_
הגישה לשדה זה מסונכרנת, אך מצד שני אין בדיקה של מיקום המכונית טרם הוספתה לרשימה, במתודה add במחלקה SimpleJunction, כך שניתן להוסיף מכונית בעלת מיקום הסותר את התכונה הנשמרת. כך שהמחלקה אינה בטוחה.
מי שציין כי המחלקה בטוחה בגלל שהגישה ל cars_ מסונכרנת, קיבל את מלוא הנקודות.

ב. יש להוסיף לתכונה הנשמרת את התנאי על צבע הרמזורים

```
(getLight(Direction.RIGHT) == GREEN  || getLight(Direction.LEFT) == GREEN ) →
  (getLight(Direction.UP) != GREEN  && getLight(Direction.DOWN) != GREEN )
&&
(getLight(Direction.UP) == GREEN  || getLight(Direction.DOWN) == GREEN ) →
  (getLight(Direction.LEFT) != GREEN  && getLight(Direction.RIGHT) != GREEN )
```

ג. המחלקה אינה בטוחה:
- בשל הסיבה שצוינה בסעיף א
- אין בקוד שום אילוץ על צבע הרמזורים, הם משנים את צבעם ע"פ הטיימר, כך שיכול להיות ששני רמזורים של כיוונים חותכים יהיו בעלי צבע ירוק.

ד. נשתמש במנגנון של OrientationSemaphore ממועד א, אך במקום לאלץ את התקדמות המכונית במתודה move במחלקה SimpleJunction, נאלץ את שינוי צבע הרמזור במתודה run במחלקה TrafficLight

מי ששם את המנגנון במתודה move קיבל בחסד חמש נקודות – זה לא מבטיח שמירה על האינווריאנטה.
<u>האינווריאנטה מאלצת כעת את צבעי הרמזורים ולא את תנועת המכוניות – זוהי נקודה מהותית!</u>

השינויים בקוד מהפתרון של מועד א מודגשים בצהוב

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

public class TrafficLight implements Runnable {

        public static TrafficLight create(Direction dir, OrientationSemaphore orientation, int redInterval, int
            yellowInterval, int greenInterval) {
                TrafficLight tl = new TrafficLight(dir,orientation, redInterval,yellowInterval,greenInterval);
                new Thread(tl).start();
                return tl;
        }
        private TrafficLight(Direction dir, OrientationSemaphore orientation, int redInterval, int yellowInterval,
```

```
            int greenInterval,) {
                _light = Light.RED;
                _redInterval = redInterval; _yellowInterval = yellowInterval; _greenInterval = greenInterval;
                _moveThreads = new HashSet<Thread>();
                _orientation = orientation;
                _dir = dir;
        }

        public synchronized void addMoveThread(Thread moveThread) {
                _moveThreads.add(moveThread);
        }

        public synchronized Light getLight() {
                        return _light;
        }

        public void run() {
                try {
                        Thread.sleep(_redInterval);
                        synchronized (this) { _light = Light.YELLOW; }
                        Thread.sleep(_yellowInterval);
                        _orientation.acquire(_dir);
                        synchronized (this) { _light = Light.GREEN; notifyAll(); }
                        Thread.sleep(_greenInterval);
                        synchronized (this) {
                                _light = Light.RED;
                                _orientation.release(_dir);
                                for (Thread moveThread : _moveThreads)
                                        moveThread.interrupt();
                                }
                } catch (InterruptedException e) {
                        return;
                }
        }

        Light _light;
        Set<Thread> _moveThreads;
        final int _redInterval, _yellowInterval, _greenInterval;
        OrientationSemaphore _orientation;
        Direction _dir;
}
```

```
enum Orientation {HORIZONTAL, VERTICAL, NONE}

class OrientationSemaphore {

	OrientationSemaphore() {
		_orientation = Orientation.NONE;
		_locks = 0;
	}

	public synchronized void acquire(Direction dir) throws InterruptedException {
		if (dir == Direction.LEFT || dir == Direction.RIGHT) {
			while (_orientation != Orientation.HORIZONTAL && _orientation != Orientation.NONE)
			  wait();
			_orientation = Orientation.HORIZONTAL;
			_locks++;
		} else {
			if (dir == Direction.UP || dir == Direction.DOWN) {
			  while (_orientation != Orientation.VERTICAL  && _orientation != Orientation.NONE)
			    wait();
			  _orientation = Orientation.VERTICAL;
			  _locks++;
			}
		}
	}

	 public synchronized void release() throws InterruptedException {
		 _locks--;
		 if (_locks == 0) {
			 _orientation = Orientation.NONE;
			 notifyAll();
		 }
	}

	private Orientation _orientation;
	private int _locks;
}
```

```
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);
        _orientation = new OrientationSemaphore();
}

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
                        try{
                                //2.2.1 wait for a legal orientation
                                _orientation.acquire(dir);
                                if (light.getLight() == Light.GREEN)
                                        try {
                                                //2.2.2 advance the car
                                                 synchronized (car) { car.advance(); }
                                                _orientation.release();
                                        } catch (CarAdvancedException e) {
                                                 _orientation.release();
                                                 throw new MoveException(e);
                                        }
                                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 and release the orientation semaphore
                                        synchronized (cars) { cars.add(0, car); }
                                        _orientation.release();
                                        return;
                                }
                        } catch (InterruptedException e) {
                                // The TrafficLight interrupts all move threads when the light turns to RED
                                // In case the light turned to RED during the waiting for legal orientation,
                                // return car back to the waiting list
                                synchronized (cars) { cars.add(0, car); }
                                return;
                        }
                }
        }
}

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

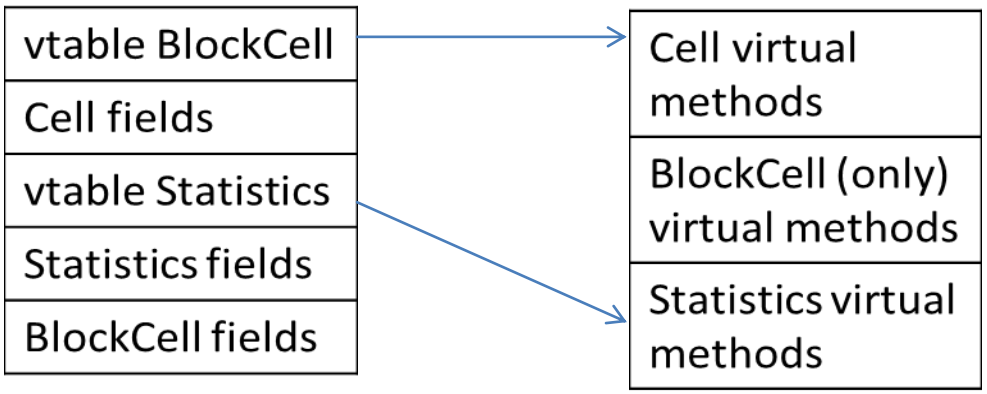
שאלה 2 (30 נקודות)

סעיף א

```
BlockScene::BlockScene(int nrows, int ncols)
{
	_nrows=nrows;
	_ncols=ncols;

	_rowscols = new Cell**[nrows];

	for (int i=0; i<nrows; i++)	{
		_rowscols[i] = new Cell*[ncols];

		for (int j=0; j<ncols; j++)		{
			_rowscols[i][j] = new BlockCell();
		}
	}
}

BlockScene::~BlockScene()
{
	for (int i=0; i<_nrows; i++){
		for (int j=0; j<_ncols; j++){
			delete _rowscols[i][j];
		}
		delete _rowscols[i];
	}
	delete _rowscols;
}


BlockScene::BlockScene(const BlockScene &other)
{
	_nrows=other._nrows;
	_ncols=other._ncols;

	_rowscols = new Cell**[_nrows];

	for (int i=0; i<_nrows; i++)	{
```

```
		_rowscols[i] = new Cell*[_ncols];

		for (int j=0; j<_ncols; j++)		{
			_rowscols[i][j] = new BlockCell(other._rowscols[i][j]);
		}
	}

}

BlockScene& BlockScene::operator=(BlockScene &other)
{
	for (int i=0; i<_nrows; i++){
		for (int j=0; j<_ncols; j++){
			delete _rowscols[i][j];
		}
		delete _rowscols[i];
	}
	delete _rowscols;



	_nrows=other._nrows;
	_ncols=other._ncols;

	_rowscols = new Cell**[_nrows];

	for (int i=0; i<_nrows; i++)  {
		_rowscols[i] = new Cell*[_ncols];

		for (int j=0; j<_ncols; j++)		{
			_rowscols[i][j] = new BlockCell(other._rowscols[i][j]);
		}
	}

	return *this;
}
```

סעיף ב



סעיף ג

תשובה 3

שאלה 3 (30 נקודות)

סעיף א

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

סעיף ב

```
String host = args[0];
int port = Integer.parseInt(args[1]);
StompConnector stompConnector = new StompConnectorImpl(host, port);
java.rmi.Naming.rebind("//132.23.5.8:2010/StompConnector", stompConnector);
```

סעיף ג

```
String login = args[0];
String passcode = args[1];
StompConnector stompConnector =
(StompConnector)java.rmi.Naming.lookup("//132.23.5.8:2010/stompConnector");
StompOperator stompOperator = stompConnector.connect(login, passcode);
stompOperator.subscribe("q1");
stompOperator.send("q3", "Suzy Se");
```

סעיף ד

גם RMI וגם STOMP מבוססים על TCP. לכן כל פעולת תקשורת בין הלקוח לבין StompConnectorRMI ובין StompConnectorRMI והשרת עוברת דרך TCP. ניתן לפרט פעולות תקשורת בדרכים שונות, למשל, לחיצת יד המשולשת של TCP יכולה להיחשב לפעולה 1 או ל3 פעולות, או לחלוטין לא להיחשב להעברת מידע מתהליך לתהליך, אלא לפעולת שירות. כל תשובה עקבית ומנומקת התקבלה.

למשל, אם נגדיר כי יצירה וניתוק חיבור אינם נחשבים לפעולה, וכי כל העברת נתונים הלוך-חזור היא פעולה אחת:

- lookup: פעולה אחת
- connect: 2 פעולות (מהלקוח למתאם, מהמתאם לשרת)
- subscribe: 2 פעולות (מהלקוח למתאם, מהמתאם לשרת)

- send: 2 פעולות (מהלקוח למתאם, מהמתאם לשרת)

סך הכל 7 פעולות.

סעיף ה

1. לא נכון, מספר פעולות תקשורת לא תלוי בניהול חישובים מקביליים בשרת
2. לא נכון, השרת אמין במידה שווה בשני המקרים (אך גם תשובה "נכון" התקבלה בהנחה שהתכוונתם כי thread-per-clientיקרוס ראשון תחת עומס)
3. נכון, אחת המטרות של - Reactor pattern ניצול יעיל יותר של המעבד (והזכרון)

שאלה 4 (10 נקודות)