גיליון תשובות

מספר נבחן:_____

שאלה 1 (30 נקודות)

סעיף א (14 נקודות)

- כפי שנלמד בהרצאות, התכונה נשמרת ותנאי התחלה וסיום מוגדרים על הממשק, ולא על מימוש ספציפי. ההגדרה צריכה להתבסס על שאילתות בסיס. במידה והשאילתות בממשק אינן מספיקות כדי להגדיר את התכונה הנשמרת והתנאים, יש להוסיף כאלה לממשק.
- לכל ממשק יש תכונה נשמרת אחת, המתייחסת למצב הנתון של מופעי הממשק, ולא לאופי פעולתם
- לכל מתודה בממשק יש תנאי התחלה וסיום, המגדירים את אופי פעולתה, ע"י תאור המצב בהתחלה ובסוף ביצוע המתודה.
- בשאלה נדרשתם לנסח את התכונה והתנאים ולא לכתוב מתודות בדיקה.

בבדיקת סעיף זה הושם דגש מיוחד על העקרונות הנ"ל

בממשק הנתון יש רק פקודות, כך שלא ניתן להגדיר שום דבר בלי מספר שאילתות. נוסיף את השאילתות הבאות כדי להפוך את הממשק לבר הגדרה:

```
// returns waiting cars in the given direction
List<Car> getCars(Direction dir);

// returns the light in the given direction
Light getLight(Direction dir);

// returns a set of directions that contradict the given direction (e.g., {LEFT,RIGHT} for UP)
Set<Direction> getOpoositeDir(Direction dir);


@INV:
for each car in getCars(Direction.UP)
  car.getPosition().equals(0,1) &&
for each car in getCars(Direction.DOWN)
  car.getPosition().equals(0,-1) &&
```

```
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;
```

סעיף ב (16 נקודות)

כפי שנלמד בכיתה, עבור מתודות של אובייקט החשוף בפני מספר ת'רדים, כמו הצומת בשאלה, אי קיום של תנאי ההתחלה גורר המתנה עד לשינוי המצב.
בנוסף, כפי שנלמד בכיתה, גישה למצב הפנימי של אובייקט החשוף למספר ת'רדים, כרוכה בסנכרון.

מכאן נובעים רכיבי המנגנון הנדרש בתשובה:

<u>סנכרון</u>:

תור המכוניות בכל כיוון עשוי להיות חשוף בפני מספר ת'רדים. נניח, ת'רד המוסיף מכוניות ע"י ביצוע add, ות'רד המזיז מכוניות ע"י ביצוע move.
גם הרמזור עשוי להיחשף פני מספ ת'רדים: הת'רד של הרמזור, ות'רד אחר המבצע move, תוך בדיקת צבע אור הרמזור.
מכאן שיש לסנכרן את הגישה לתור המכוניות ואת הגישה לרמזור

המתנה:
במתודה move יש כמה תנאי התחלה להמתנה:
- המתנה לאור ירוק
- המתנה למכונית בתור המכוניות
- המתנה לאור אדום בכיוונים הנגדיים

סיום:
- במידה והרמזור מתחלף לאדום המתודה מסתיימת. יש לזהות תרחיש זה, בפרט כאשר הת'רד מחכה על תנאי אחר

בבדיקת התשובות, הושם דגש רק על הדברים הבסיסיים ולא על ה'לוגיקה':
- המתנה לאור ירוק, ולתור שאינו ריק
- סנכרון הרמזור והמכוניות

בפיתרון ממומש מנגנון של סמפור OrientationSemaphore המטפל בדו כיווניות.
לא היה מצופה שתממשו דבר כזה, כל אמירה על סנכרון כלשהו זיכתה בנקודה. מאידך לא ירדו לנקודות למי שלא מימש זאת.

כמובן שלא ירדו נקודות על תחביר, אי ידיעת map, list, מי שציין שהוא מניח ש List מתנהג כ BlockingQueue תשובתו נתקבלה, וכו'.

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

public 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;
		_moveThreads = new HashSet<Thread>();
	}

	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);
			synchronized (this) { _light = Light.GREEN; notifyAll(); }
			Thread.sleep(_greenInterval);
			synchronized (this) {
				_light = Light.RED;
				for (Thread moveThread : _moveThreads)
					moveThread.interrupt();
				}
		} catch (InterruptedException e) {
			return;
		}
	}

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

```
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;
}
```

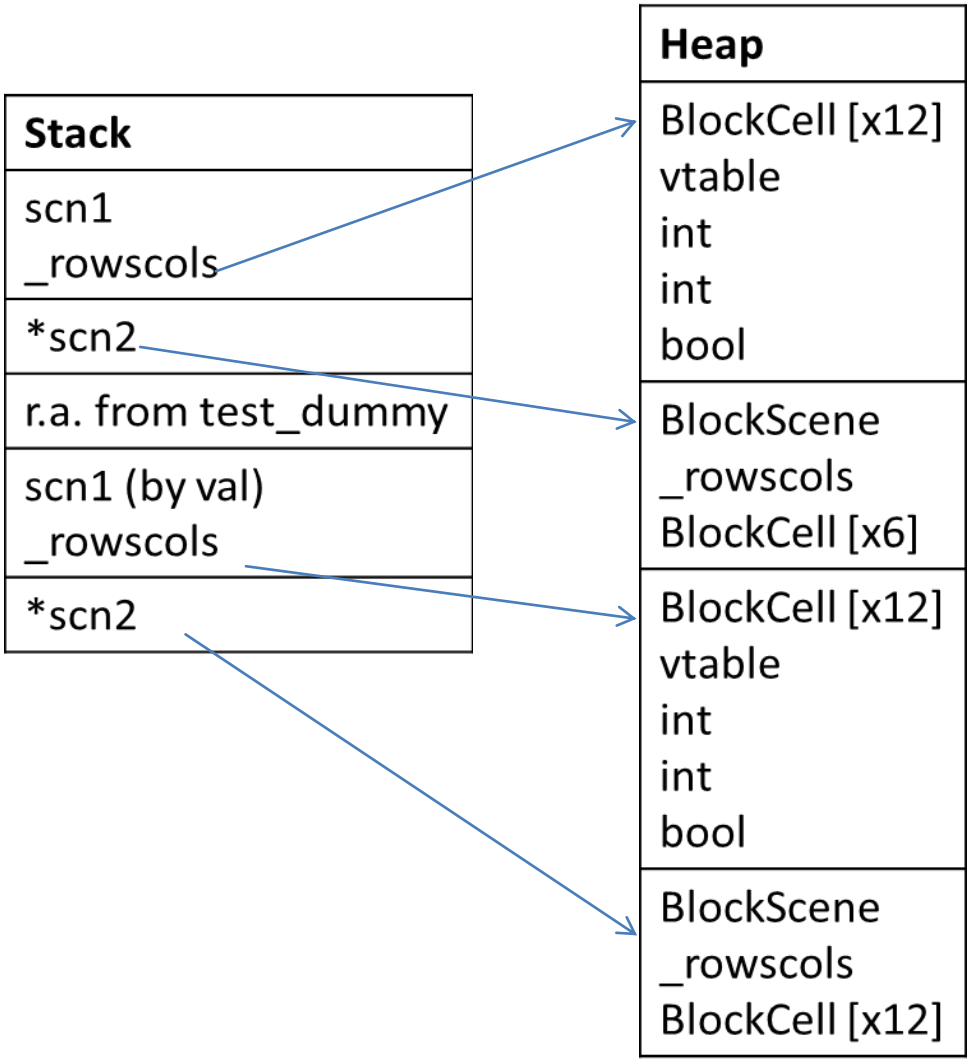
שאלה 2 (30 נקודות)

סעיף א (12 נקודות)

```
BlockScene::BlockScene(int nrows, int ncols){
	_rowscols.resize(nrows);
	for (std::vector<std::vector <Cell*>>::iterator row_cit = _rowscols.begin(); row_cit != _rowscols.end();
++row_cit){
		(*row_cit).resize(ncols);
		for (std::vector <Cell*>::iterator col_cit = (*row_cit).begin(); col_cit != (*row_cit).end(); ++col_cit){
			(*col_cit) = new BlockCell();
		}
	}
}

BlockScene::~BlockScene(){
	for (std::vector<std::vector <Cell*>>::iterator row_cit = _rowscols.begin(); row_cit != _rowscols.end();
++row_cit){
		for (std::vector <Cell*>::iterator col_cit = (*row_cit).begin(); col_cit != (*row_cit).end(); ++col_cit){
			delete (*col_cit);
		}
		(*row_cit).clear();
	}
	_rowscols.clear();
}


BlockScene::BlockScene(const BlockScene &other){
	int i,j;
	_rowscols.resize(other.getNRows());

	i=0;
	for (std::vector<std::vector <Cell*>>::iterator row_cit = _rowscols.begin();
		row_cit != _rowscols.end(); ++row_cit, i++)	{
		(*row_cit).resize(other.getNCols());
		j=0;
		for (std::vector <Cell*>::iterator col_cit = (*row_cit).begin();
			col_cit != (*row_cit).end(); ++col_cit, j++)		{
			(*col_cit) = new BlockCell(other.getCell(i,j));
		}
	}
}
BlockScene& BlockScene::operator=(BlockScene &other){
```

```
	int i,j;
	for (std::vector<std::vector <Cell*>>::iterator row_cit = _rowscols.begin(); row_cit != _rowscols.end();
++row_cit){
		for (std::vector <Cell*>::iterator col_cit = (*row_cit).begin(); col_cit != (*row_cit).end(); ++col_cit){
			delete (*col_cit);
		}
		(*row_cit).clear();
	}
	_rowscols.clear();
	_rowscols.resize(other.getNRows());

	i=0;
	for (std::vector<std::vector <Cell*>>::iterator row_cit = _rowscols.begin();
		row_cit != _rowscols.end(); ++row_cit, i++)	{
		(*row_cit).resize(other.getNCols());
		j=0;
		for (std::vector <Cell*>::iterator col_cit = (*row_cit).begin();
			col_cit != (*row_cit).end(); ++col_cit, j++)	{
			(*col_cit) = new BlockCell(other.getCell(i,j));
		}
	}
	return *this;
}
```

סעיף ב (13 נקודות)



סעיף ג (5 נקודות) 3. אין שחרור זכרון ב heap

שאלה 3 (30 נקודות)

סעיף א (20 נקודות)

הצעה ראשונה

בסעיף זה ניתן גם להעביר פורט נוסף כפרמטר לבנאי של הריאקטור, או לבחור בו בדרך אחרת. הפתרון המוצע שומר על הממשק הקיים של המחלקה.

```
diff --git a/Reactor/src/reactor/Reactor.java b/Reactor/src/reactor/Reactor.java
index 887b470..a9d8d67 100644
--- a/Reactor/src/reactor/Reactor.java
+++ b/Reactor/src/reactor/Reactor.java
@@ -95,2 +95,3 @@ public class Reactor implements Runnable {
                        ServerSocketChannel ssChannel = null;
+                       ServerSocketChannel ssChannelx = null;

@@ -99,2 +100,3 @@ public class Reactor implements Runnable {
                                ssChannel = createServerSocket(_port);
+                               ssChannelx = createServerSocket(_port+1);
                        } catch (IOException e) {
@@ -106,2 +108,3 @@ public class Reactor implements Runnable {
                        ConnectionAcceptor connectionAcceptor = new ConnectionAcceptor( ssChannel, _data);
+                       ConnectionAcceptor connectionAcceptorx = new ConnectionAcceptor( ssChannelx,
_data);

@@ -112,2 +115,3 @@ public class Reactor implements Runnable {
                                ssChannel.register(selector, SelectionKey.OP_ACCEPT, connectionAcceptor);
+                               ssChannelx.register(selector, SelectionKey.OP_ACCEPT,
connectionAcceptorx);
                        } catch (ClosedChannelException e) {
```

הצעה שניה

קיימים פתרונות שונים. כל פתרון שהסתמך על תרדים מקביליים, תרד לכל סלקטור, עם executor משותף, התקבל.

```
diff --git a/Reactor/src/reactor/Reactor.java b/Reactor/src/reactor/Reactor.java
index 887b470..53ad9a6 100644
--- a/Reactor/src/reactor/Reactor.java
+++ b/Reactor/src/reactor/Reactor.java
@@ -90,5 +90,17 @@ public class Reactor implements Runnable {
                */
+
                public void run() {
                        // Create & start the ThreadPool
```

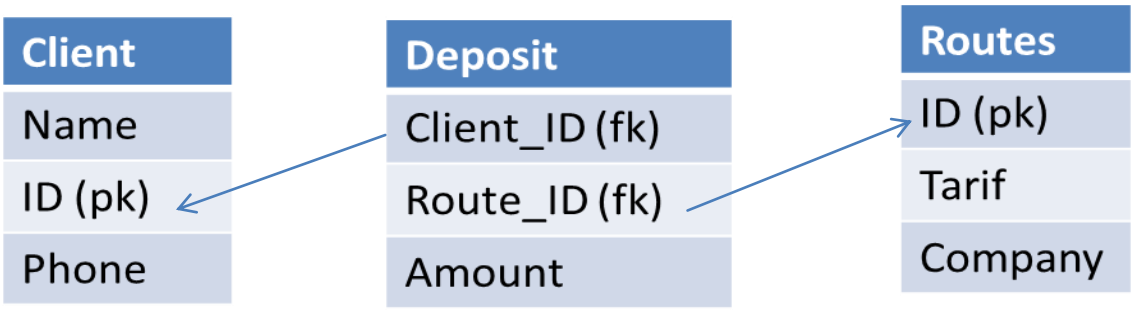
```
-                       ExecutorService executor = Executors.newFixedThreadPool(_poolSize);
+                       final ExecutorService executor = Executors.newFixedThreadPool(_poolSize);
+
+                       Thread rx = new Thread(new Runnable() { public void run() { _run(executor, _port+1); } });
+
+                       rx.start();
+                       _run(executor, _port);
+                       try {
+                               rx.join();
+                       } catch(InterruptedException e) {}
+               }
+
+               public void _run(ExecutorService executor, int _port) {
                       Selector selector = null;
```

סעיף ב (10 נקודות)

שתי ההצעות מייעלות את ההתחברות פי שניים: במעבר על הפורטים המוכנים לפעולות קלט\פלט\התחברות יש סיכוי כפול, לעומת server socket אחד, להתחברות ביחס לפעולות קלט\פלט.
בנוסף, בפתרון השני פעולות על הערוצים בשני הסלקטורים יכולות להתבצע במקביל, אך שיפור זה זניח, כי פעולות קלט\פלט\התחברות על אותה כתובת השרת מסונכרנות בסביבת זמן ריצה. מהצד השני, הפתרון השני צורך משאבים נוספים כגון זכרון שנדרשים לתרד נוסף.

שאלה 4	(10 נקודות)

סעיף א (5 נקודות)



סעיף ב (5 נקודות)

```
SELECT Deposit.Amount
FROM Deposit INNER JOIN Client ON  Deposit.Client_ID=Client.ID
        INNER JOIN Routes ON Deposit.Route_ID=Routes.ID
WHERE Client.Name = 'Harpo Marx' AND Routes.Company = 'Galei HaMa'avir'
```