גיליון תשובות

מספר נבחן:_____

שאלה 1 (30 נקודות)

א.

```
@inv getRectBottom()  ≤ getBallY() ≤ getRectUpper() &&
getRectLeft()  ≤ getBallX() ≤ getRectRight()
public boolean checkInv() {
return (getRectBottom()  ≤ getBallY() ≤ getRectUpper() &&
getRectLeft()  ≤ getBallX() ≤ getRectRight());
}

Up:
@pre getBallY() < getRectUpper()
@post @pre(getBallY() + 1) = getBallY()
public boolean checkPreCondUp() {
        return (getBallY() < getRectUpper());
}
Down:
@pre getRectBottom() <  getBallY()
@post @pre(getBallY() - 1) = getBallY()
public boolean checkPreCondDown() {
        return (getRectBottom() <  getBallY());
}

Left:
@pre  getRectLeft() < getBallX()
@post @pre(getBallX() - 1) = getBallX()
public boolean checkPreCondLeft() {
        return (getRectLeft() < getBallX());
}
Right:
@pre getBallX() < getRectRight()
@post @pre(getBallX() + 1) = getBallX()
public boolean checkPreCondRight() {
        return (getBallX() < getRectRight());
}
```

ב. משמעותה של תוכנית בטוחה הינה תוכנית בה נשמרת ה-inv של מחלקת SimpleBlockedBall, כפי שהוגדרה בסעיף א'. מכיוון ששני האוביקטים האקטיביים פועלים באופן בלתי תלוי (ציר אופקי וציר אנכי), הכדור

(שהינו משאב משותף, לכאורה) "מופרד" לשני אוביקטים שאינם תלוים אחד בשני, כך שלא יתכן מצב בו הכדור יוצא מהמסגרת.

ג. כעת התוכנית אינה בטוחה. נסמן את הת'רדים:

T1 – Horizontal Movement

T2 – Vertical Movement

T3 – Wind

נתבונן במשפחת הריצות/תסריטים הבאה:
- הכדור נמצא מרחק צעד אחד מהמסגרת.
- T1/T2 מריץ פעולת הזזה לעבר המסגרת ומספיק לבצע את בדיקת תנאי הקדם, שכאמור תחזיר תשובה חיובית שכן הכדור נמצא צעד אחד מהמסגרת.
- T3 מריץ את אותה הפעולה המופעלת ע"י T1/T2. הוא עובר את בדיקת תנאי הקדם של הפעולה ומבצע את ההזזה. כעת הכדור נמצא על המסגרת, כלומר כל תנועה נוספת לעבר אותו הכיוון תוציא אותו החוצה ותסתור את ה-inv של המחלקה.
-T1/T2 מקבלים זמן ריצה ומבצעים את הזזת הכדור, שכן תנאי הקדם נבדק כבר קודם. כעת הכדור נמצא מחוץ למסגרת.

תשובה נוספת אשר התקבלה, על אף שאינה סותרת את ה-inv:
הכדור נמצא בנקודה כלשהי, T2 מפעיל פעולת תנועה ימינה, כאשר הוא עובר את תנאי הקדם, קורא את ערך ה-y של הכדור ומבצע מקומית חישוב של הערך העדכני של y, אך טרם הספיק לכתוב את הערך לזיכרון המשותף (כלומר, T1 ו-T3 אינם מבחינים בפעולתו של T2).
כעת T1 ו-T3 מבצעים מספר פעולות הזזה (למשל 4 הזזות ימינה של הכדור), ולאחר מכן T2 משלים את פעולתו וכותב לזכרון המשותף את הערך אשר חושב קודם. נשים לב כי הערך אשר נכתב ע"י T2 הינו ערך שאינו עדכני הדורס את 4 ההזזות ימינה.
תסריט זה מוכיח שהמחלקה אינה לינארנבילית, מכיוון שבוצעו 5 פעולות הזזה ימינה, בעוד שבתום הריצה הכדור זז צעד יחיד ימינה.

ד. על מנת לאפשר שימוש בטוח אך מקבילי (מבחינת הצירים עליהם מבוצעות הפעולות), נמיר את השדות _ballX ו-_ballY מטיפוס int ל-Integer וזאת על מנת שיהיו אובייקטים ולא פרימיטיביים, שכן לאובייקטים יש מפתח המצורף אליהם, עליו ניתן לבצע synchronized.
השינויים הדרושים במתודות של המחלקה:

```
public int getBallX() {
  synchronized(_ballX){
    return _ballX;
  }
}

public int getBallY() {
  synchronized(_ballY){
    return _ballY;
  }
}

public void up() throws Exception {
synchronized(_ballY){
if (!checkPreCondUp())
        throw new Exception("The pre-condition for the up() method does not hold!");
```

```
_ballY++;
}
}

public void down() throws Exception {
synchronized(_ballY){
if (!checkPreCondDown())
        throw new Exception("The pre-condition for the down() method does not hold!");
_ballY--;
}
}





public void right() throws Exception {
synchronized(_ballX){
if (!checkPreCondRight())
        throw new Exception("The pre-condition for the right() method does not hold!");
_ballX++;
}
}

public void left() throws Exception {
synchronized(_ballX){
if (!checkPreCondLeft())
        throw new Exception("The pre-condition for the left() method does not hold!");
_ballX--;
}
}
```

<u>פתרונות אפשריים נוספים</u>:

א. שימוש בשני Semaphore-ים, כאשר באחד מהם נעשה שימוש ב-up/down ובאחר נעשה שימוש ב-left/right. יש לשים לב שביצוע ה-acquire הינו הפעולה הראשונה המתבצעת במתודה, ואילו release הינה הפעולה האחרונה, כלומר הן עידכון ערך ה-x/y והן בדיקת תנאי הקדם מבוצעות בתוך ה-Semaphore.
ב. שימוש באובייקט יחיד המשמש כמפתח עבור פעולות up/down והגדרת הפעולות left/right כ-synchronized בחתימת הפונקציות הללו. המפתח הנרכש עבור הכדור שונה מהמפתח הנרכש עבור האובייקט הנוסף, ולכן אין מניעה של פעולה מקבילית על הצירים השונים.

שאלה 2	(30 נקודות)

סעיף א (12 נקודות)

ראשית סליחה על הדו ממשמעות שבבחירת הפרמטר height, אשר כוונתו הייתה עומק הרקורסיה. מי שלא שמע את העדכון בזמן המבחן והתיחס לגובה המלבן וחילק לפי גובה המלבן – קיבל את רוב הנקודות.

```
void RectangleCell ::subdivide(int height)
{
        If (height=0)
                return;
        height --;
        float sizeX = (_right - _left) /2;
        float sizeY = (_top - _bottom) /2

        _topLeftSon = new RectangleCell(_left, _bottom + sizeY, _right-sizeX, _top);
        _topLeftSon ->subdivide(height);

        _topRightSon = new RectangleCell(_left+sizeX, _bottom+sizeY, _right, _top);
        _topRightSon ->subdivide(height);

        _bottomLeftSon = new RectangleCell(_left, _bottom, _right-sizeX, _top-sizeY);
        _bottomLeftSon ->subdivide(height);

        _bottomRightSon = new RectangleCell(_left+sizeX, _bottom, _right, _top-sizeY);
        _bottomRightSon ->subdivide(height);
}


SceneCell* RectangleCell ::retrieveCell(float x, float y)
{
        if (x<_left || x>_right || y<_bottom || y> _top)
                return NULL;
        SceneCell *res;

        if (_topLeftSon){
                res=_topLeftSon->retrieveCell(x,y);
                if (res)    return res;
                }
        if (_topRightSon){
                res=_topRightSon->retrieveCell(x,y);
                if (res)    return res;
                }
        if (_bottomLeftSon){
                res=_bottomLeftSon->retrieveCell(x,y);
                if (res)    return res;
                }
```

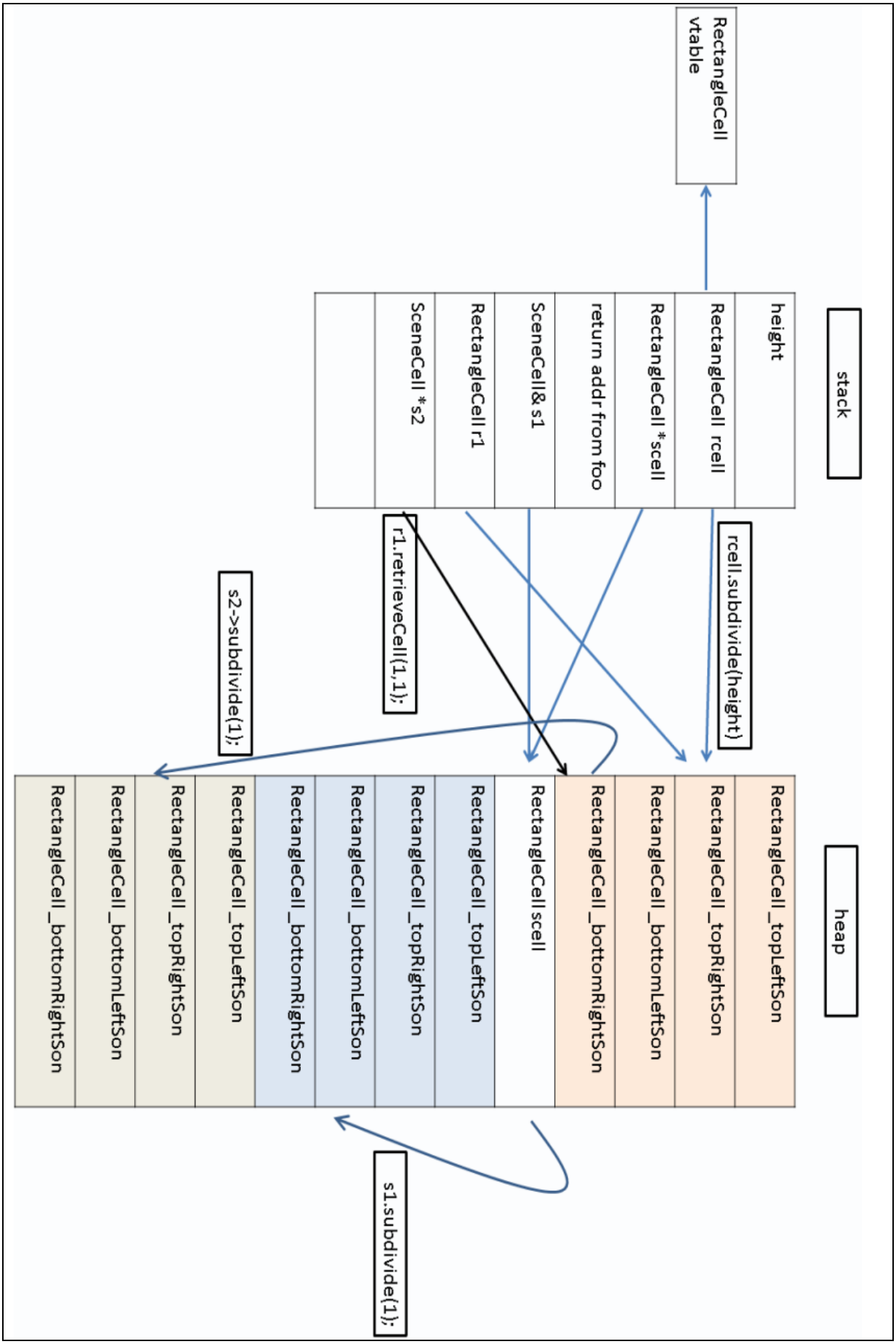
```
        if (_bottomRightSon){
                res=_bottomRightSon ->retrieveCell(x,y);
                if (res)    return res;
                }
                return this;

}
```

סעיף ב (6 נקודות)

הבעיה היא שלא מוגדר destructor עמוק כך שביציאה מmain יישאר זיכרון שלא ישוחרר. הפתרון הוא הגדרת destructor עמוק. מי שענה לגבי עומק 17 שצורך יותר מידי זיכרון או לחלוקת גובה המלבן 17 פעמים (שוב בגלל הבלבול בשמות) קיבלו נקודות.

```
virtual RectangleCell ::~RectangleCell()
{
        delete _topLeftSon;
        delete _topRightSon;
        delete _bottomLeftSon;
        delete _bottomRightSon;
}
```

סעיף ג (12 נקודות)


stack
heap
height
RectangleCell rcell
RectangleCell *scell
return addr from foo
SceneCell& s1
RectangleCell r1
SceneCell *s2
RectangleCell vtable
rcell.subdivide(height)
r1.retrieveCell(1,1);
s2->subdivide(1);
s1.subdivide(1);
RectangleCell_topLeftSon
RectangleCell_topRightSon
RectangleCell_bottomLeftSon
RectangleCell_bottomRightSon
RectangleCell scell
RectangleCell_topLeftSon
RectangleCell_topRightSon
RectangleCell_bottomLeftSon
RectangleCell_bottomRightSon
RectangleCell_topLeftSon
RectangleCell_topRightSon
RectangleCell_bottomLeftSon
RectangleCell_bottomRightSon

שאלה 3 (20 נקודות)

סעיף א (5 נקודות)

מאחר וההודעות מכילות יותר מ 12 בתים, יתכן ובכל קריאה מה socket במתודת ה ()read ב ConnectionHandler נקרא רק בית אחד בכל פעם. כל קריאה של בתים במתודת ה ()read מלווה בהוספת ה task של הלקוח לתור המשימות של ה Executor (= ביצוע execute עם task_), כך שיתכנו שנים עשר עותקים של ה task של הלקוח בתור המשימות של ה Executor.

ב Executor יש שנים עשר ת'רדים. אם כל אחד משך משימה מהתור, יעבוד כל אחד מהם על אותו אובייקט – ה task של אחד הלקוחות. הת'רד הראשון שיכנס למתודת ה ()run המסונכרנת, ינעל את האובייקט ולא יאפשר לשאר הת'רדים, הנדרשים לאותו מנעול לבצע את מתודת ה ()run של אותו אובייקט, לעבוד. כך שהם יכנסו למצב blocked, על אף שיש משימות נוספות לביצוע בתור עבור הבתים שהתקבלו מהלקוח השני.

סעיף ב (5 נקודות)

הצעת הסטודנט תאפשר שחרור של המנעול בכל סיבוב של הלולאה, כך שת'רד אחר יוכל לתפוש את המנעול ולרוץ, אך עדיין רק ת'רד אחד יכול לרוץ בכל פעם עקב סנכרון הקוד בלולאה.

סעיף ג (14 נקודות)

כדי להבטיח מופע אחד בלבד של כל task בתור המשימות של ה Executor, נבדוק לפני הוספתו לתור - במתודת ה ()read של ה ConnectionHandler – האם הוא קיים כבר בתור.

הגישה לתור המשימות של ה executor ניתנת על ידי המתודה ()getQueue במחלקה ThreadPoolExecutor - המימוש הנתון של ExecutorService – המחזירה את תור המשימות כאובייקט מסוג BlockingQueue<Runnable>.

<u>הערה</u>: השאלה התייחסה לתור המשימות (=אלו שעדיין לא נלקחו ע"י אחד הת'רדים ב thread pool של ה executor לביצוע), תשובות שטיפלו רק במשימות שנלקחו ומבוצעות כבר על ידי הת'רדים קיבלו רק חלק מהנקודות..

```
public class ConnectionHandler<T> {
  ...

  public void read() {
    if (_protocol.shouldClose())
      return;

    SocketAddress address = _sChannel.socket().getRemoteSocketAddress();
    logger.info("Reading from " + address);

    ByteBuffer buf = ByteBuffer.allocate(BUFFER_SIZE);
    int numBytesRead = 0;
```

```
    try {
        numBytesRead = _sChannel.read(buf);
    } catch (IOException e) {
        numBytesRead = -1;
    }
   if (numBytesRead == -1) {
        logger.info("client on " + address + " has disconnected");
        closeConnection();
        _protocol.connectionTerminated();
        return;
    }
     buf.flip();
    _task.addBytes(buf);
    if (!execute.getQueue().contains(_task))
      _data.getExecutor().execute(_task);
}
```

שאלה 4 (10 נקודות)

א: לא נכון
ב: לא נכון
ג: נכון [הממשק Remote מרחיב את הממשק Serializable. אין מדובר בהתחכמות, אלא בהבנה בסיסית שלא ניתן להעביר אובייקט דרך תשתית הרשת אם הוא אינו בר המרה ושחזור ל/מ bytes]
ד: נכון
ה: לא נכון

שאלה 5 (10 נקודות)

סעיף א (5 נקודות)

```
CREATE TABLE doc (
   id INTEGER PRIMARY KEY,
   subject VARCHAR(30),
   link VARCHAR(30),
   authorId INTEGER);

ALTER TABLE doc
   ADD FOREIGN KEY (authorId)
   REFERENCES author(id);

CREATE TABLE author (
   id INTEGER PRIMARY KEY,
   name VARCHAR(30),
   birthdate VARCHAR(30),
   address VARCHAR(30));
```

סעיף ב (5 נקודות)

```
SELECT doc.link, author.name, author,date, author.address
   FROM doc LEFT OUTER JOIN author
   ON doc.authorId = author.id
   WHERE doc.subject="פיוט"
   ORDER BY doc.id;
```