גיליון תשובות מספר נבחן:_____

על תשובות ריקות לשאלות הפתוחות יינתן 20% מהניקוד!

שאלה 1 (30 נקודות)

א - 1
ב - 5
ג – 2
[הערה: מסיח (1) התגלה כבעייתי בדיעבד לאחר הבחינה. ניתן לחשוב שהיה אמור להיקרא copy constructor אבל בפועל נקרא רק constructor כאופטימיזציית קומפיילר. נא להתעלם ממקרה זה]
ד - 4
ה - 5

שאלה 2 (30 נקודות)

להלן פתרון שאלה 2 - מומלץ לעיין לפני שמגישים ערעור.

<u>סעיף א</u>

- אינווריאנטה

נוספה אפשרות למחסנית בגודל שאינו חסום, כך שיש לבטא זאת בתכונה הנשמרת.
כפי שנלמד בכיתה, אם לא ניתן לבטא הבט מסויים של מצב הממשק בשאילתות הקיימות, יש להוסיף שאילתות מתאימות. במקרה זה נוסיף את השאילתא IsLimited() המציינת האם המחסנית במוד של גודל חסום.
כעת ניתן להגדיר את התכונה הנשמרת:

```
(0 <= size() && (!isLimited() || size() <= capacity())
```

- תנאי התחלה וסיום

תנאי ההתחלה והסיום מגדירים את הסמנטיקה (המשמעות) של כל אחת מהמתודות. בפרט, בממשק שלנו, הם אמורים לבטא את משמעות הפעולות push,pop במחסנית (השינוי בגודל המחסנית אינו מספיק לבטא זאת)
גם כאן, כדי לבטא משמעות זו, יש להוסיף אם נדרש שאילתות. במקרה שלנו נדרשת שאילתא המחזירה את התוכן של אחד האברים במחסנית itemAt.
כך ניתן להגדיר את תנאי הסיום ע"פ:

- השינוי בגודל המחסנית
- הימצאות הפרמטר בראש המחסנית / החזרת הערך מראש המחסנית
- אי שינוי שאר האיברים במחסנית

<u>push</u>

```
//@PRE (!isLimited() || size() < capacity())
//@POST size() == @pre(size())+1   &&
        @param data == itemAt(size()-1)   &&
```

```
for 0 <= i < size()-1: itemAt(i) == @pre(itemAt(i))
```

pop

```
//@PRE size() > 0
//@POST size() == @pre(size())-1   &&
        @ret == @pre(itemAt(size()-1))  &&
        for 0 <= i < size()-2: itemAt(i) == @pre(itemAt(i))
```

הערות:

- הסתפקות בשינוי הגודל התנאי הסיום, מבלי להתייחס לסמנטיקה הבסיסית של push/pop גררה הורדה של נקודה אחת בכל מתודה (לפנים משורת הדין).
- התייחסות לשני המקרים בתכונה הנשמרת מבלי להגדיר את המתודה isLimited גררה הורדה של נקודה.

סעיף ב

בסעיף זה נדרש למעשה לממש את תבנית wait/notify, שנלמדה בפירוט בכיתה ובתרגולים, מבלי להשתמש ב synchronized.
כדי להקל עליכם, קוד המחסנית שקיבלתם עוצב באופן המבליט את החלופה של הנעילה והשחרור:
- נעילה: במקום synchronized, לולאת while הממתינה ב busy wait לעדכון ערך במשתנה בוליאני אטומי.
- שחרור: עדכון ערך המשתנה הבוליאני האטומי.
בנוסף, כדי למנוע מצב שסטודנט/ית יפסידו את השאלה כי לא יעלו בדעתם את מנגנון sleep/interrupt, נרמז באופן שקוף בחומר העזר על מנגנון זה (ספציפית, צריך באמת את המתודה sleepIfNotInterrupt כדי להבטיח שהת'רד לא ילך לישון אם בוצע אינארפט ע"י ת'רד אחר לאחר שחרור הנעילה).

בתשובה היה צריך לכלול את הרכיבים הבאים (המרכיבים את מנגנון wait/notify כפי שנלמד בכיתה ובתרגולים):

- בדיקה, תחת נעילה, של תנאי ההתחלה עם לולאת while
- אם התנאי אינו מתקיים
  - שחרור הנעילה
  - מעבר למצב blocked
  - תפיסה מחודשת של הנעילה
- לאחר ביצוע הפעולה, החזרה, תחת נעילה, של הת'רדים הממתינים למצב runnable.

חסרון של כל אחד מרכיבים אלו הופך את המנגנון לשגוי, כפי שנלמד בכיתה.

להלן הקוד – ההוספות לקוד מסומנות במרקר צהוב:

```
class LinkedStack<T> implements Stack<T> {

    private volatile Link<T> head;
    private volatile int size;
    private final int capacity;
    private AtomicBoolean isFree;
```

```
Set<Thread> waitingThreads;

LinkedStack(int capacity) throws Exception {
	if (capacity < 1)
		throw new Exception("Illegal capacity: " + capacity);
	this.capacity = capacity;
	this.head = null;
	this.size = 0;
	this.isFree = new AtomicBoolean(true);
	this.waitingThreads = new HashSet<Thread>();
}

public void push(T data) {
	while (!isFree.compareAndSet(true,false));

	while (size == capacity) {
		try {
			waitingThreads.add(Thread.currentThread());
			isFree.set(true);
			Thread.currentThread().sleepIfNotInterrupted();
		} catch (InterruptedException e) {
		}
		while (!isFree.compareAndSet(true,false));
	}

	head = new Link<>(head, data);
	size++;

	for (Thread t : waitingThreads)
		t.interrupt();
	waitingThreads.clear();

	isFree.set(true);
}

public T pop() {
	while (!isFree.compareAndSet(true,false));

	while (size == 0) {
		try {
			waitingThreads.add(Thread.currentThread());
			isFree.set(true);
			Thread.currentThread().sleepIfNotInterrupted();
		} catch (InterruptedException e) {
		}
		while (!isFree.compareAndSet(true,false));
```

```
        }

        T ret = null;
        if (size > 0) {
            ret = head.data;
            head = head.next;
            size--;
        }

        for (Thread t : waitingThreads)
                t.interrupt();
        waitingThreads.clear();

        isFree.set(true);

        return ret;
    }

    public int size () {
        int ret;
        while (!isFree.compareAndSet(true,false));
        ret = size;
        isFree.set(true);
        return ret;

    }

    public int capacity () {
        return capacity;
    }

}
```

סעיף ג

נדרש היה לתאר תרחיש של חבק הנובע ממעבר של ת'רדים למצב blocked בשל המתנה לשינוי תנאי התחלה.
תרחיש זה נלמד בכיתה.

הערה: בתרחיש צריך היה לתאר איזשהו הבט של הדדיות. כלומר זה לא סתם שהת'רדים נכנסים למצב blocked, אלא שהקוד שלהם תלוי אחד בשני (כמו בדוגמא בכיתה: הם מוציאים ומכניסים אברים באופן מעגלי לשמי התורים). לפנים משורת הדין, הקלתי מאוד וקיבלתי כל אזכור לכך ש'אין ת'רדים אחרים'. אם לא היתה כל התייחסות להדדיות או לעובדה שאין ת'רדים אחרים ירדה נקודה אחת בלבד.
תשובות שתיארו חבק הנובע מסנכרון/תפיסת-משאבים הן שגויות - עבור סנכרון אין מעבר למצב blocked בקוד הנוכחי, כי משתמשים ב cas.

שאלה 3	**(30 נקודות)**

שאלה 4	**(10 נקודות)**

א - 1
ב - 5