גיליון תשובות

מספר נבחן:_____

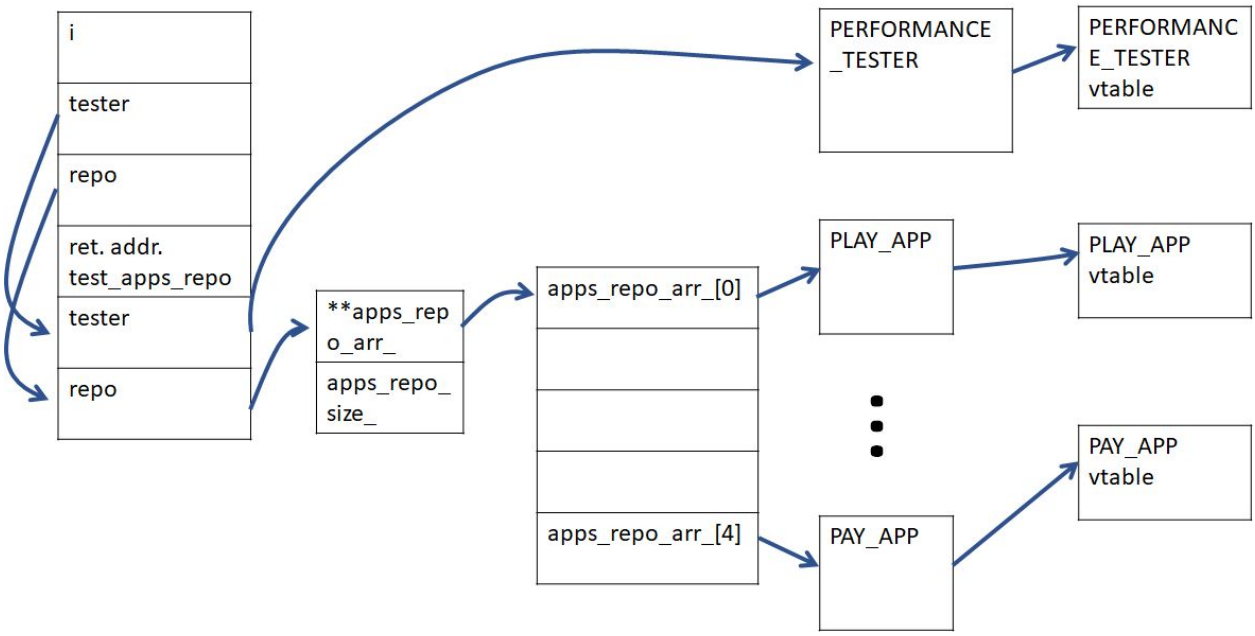
שאלה 1	(30 נקודות)

סעיף א:

```
class PAY_APP : public APP{
	virtual void testMe(TESTER *tester) { tester->test(this); }
};
class PERFORMANCE_TESTER: public TESTER
{
	virtual void test(APP *app) { app->testMe(this); }
};
```

מי שעשה CASTING איבד בין 5-10 נק' תלוי בנכונות הפתרון.
מי ששינה את הקוד לא במקומות המסומנים לא קיבל נקודות
מי שלא עבד נכון עם פוינטרים ו this איבד בין 2-6 נק'

סעיף ב:



מי ששכח vtable איבד בין 2-4 נק'
מי שלא צייר נכון את האוביקטים בheap בין 2-4 נק
מי שלא צייר נכון פרמטרים לפונק וכתובת חזרה בין 2-4 נק

<u>סעיף ג</u>:

```
virtual ~APP() { ; }
virtual ~PAY_APP() { ; }
virtual ~PLAY_APP() { ; }

MY_APPS_REPO::~MY_APPS_REPO()
{
	for (int i = 0; i < apps_repo_size_; i++)
		delete apps_repo_arr_[i];

	delete apps_repo_arr_;
}
```

מי שלא כתב שיחרור עמוק של זכרון -3
מי שלא כתב virtual בין 2-3

שאלה 2 (30 נקודות)

סעיף א:

נוסיף לממשק את שאילתא המקבלת ערך ומחזירה את הקודקוד בעל ערך זה בעץ. אם הערך אינו מופיע בעץ מוחזר null.

```
Tree<T> get(T data);
```

תנאי התחלה וסיום למתודה insert

```
//@PRE: @param data != null
//@POST: get(data) != null
```

[אם הולכים עד הסוף, צריך להוסיף לתנאי הסיום ששאר העץ לא השתנה]

תנאי התחלה וסיום למתודה remove

```
//@PRE: @param data != null
//@POST: get(data) == null ||
         (get(data).getLeft() == null && get(data).getRight() == null)
```

[אם הולכים עד הסוף, צריך להוסיף לתנאי הסיום ששאר העץ לא השתנה]

תנאי הסיום נשמרים בריצה מקבילית כי המחלקה מסונכרנת סינכרון מלא.

סעיף ב:

מתודת sum צריכה להחזיר סכום של עץ שהיה קיים בנקודת זמן כלשהי. כדי להקל על ההגדרה נצמצם את הדרישה להחזרת סכום העץ שהיה בשלב תנאי ההתחלה.

```
public int sum();
//@PRE: none
//POST: @ret == @pre(getLeft() == null ? 0 : sum(getLeft()) +
                @pre(getData()) +
                @pre(getRight() == null ? 0 sum(getRight()))
```

מאחר ו sum אינה מסונכרנת, ייתכן כי בזמן המעבר על הקודקודים יתווספו או יוסרו קודקודים אחרים במקביל ע"י ת'רד אחר. כך שהסכום לא יתייחס לעץ המקורי בשלב תנאי ההתחלה.

<u>סעיף ג</u>:

<u>אפשרות א</u>: Optimistic Try & Fail בעזרת Versioned Iterator

```
class SortedBinaryIntTree extends SortedBinaryTreeImpl<Integer>
                          implements Iterable<Integer> {

    protected int version;

    SortedBinaryIntTree(Integer data) throws Exception {
       super(data);
       version = 0;
    }

    public synchronized int getVersion() { return version; }

    public synchronized void remove(T data) throws Exception {
       super.remove(data);
       version++;
    }

    public synchronized void insert(T data) throws Exception {
        super.insert(data);
        version++;
    }

    public int sum() {
        try {
            int ret =0;
            Iterator<Integer> it = iterator();
            while (it.hasNext())
```

```
                ret += it.next();
            return ret;
        } catch (Exception e) {
            return sum();
        }
    }

    public Iterator<Integer> iterator() {
        return new Iterator<Integer> () {

            private int origVersion = getVersion();

            SortedBinaryTree<Integer> currNode = SortedBinaryIntTree.this;
            Stack<SortedBinaryTree<Integer>> stack =
                                    new Stack<SortedBinaryTree<Integer>>();

            public Integer next() {
                synchronized (SortedBinaryIntTree.this) {
                   if (origVersion != getVersion())
                       throw new Exception ("Concurrent modification!");
                    while (currNode != null) {
                        stack.push(currNode);
                        currNode = currNode.getLeft();
                    }
                    currNode = stack.pop();
                    Integer ret = currNode.getData();
                    currNode = currNode.getRight();
                    return ret;
                }
            }

            public boolean hasNext() {
                return (!stack.isEmpty() || currNode != null);
            }
        };
    }
}
```

אפשרות ב: תחזוק שדה sum - הגדלתו במתודה insert והקטנתו במתודה remove - והחזרה מסונכרנת שלו במתודת sum.

שאלה 3 (25 נקודות)

<u>סעיף א</u>:

(1)

1. פתרון שאינו סדרתי - איבד כל הנקודות
2. נעילה של-actor שבה יש הפעלה של run, איבד 2 נקודות
3. פתרון ללא נעילה, איבד 2 נקודות
4. פתרון שאין בו את ה-Runnable הראשוני - איבד 2 נקודות
5. פתרון שלא מוחק את ה-actor בסוף מ-playingNow, איבד 2 נקודות

```
private void execute(Runnable r, T actor) {
  threads.submit(() -> {
    do {
      try {
        r.run();
      } finally {
        r = complete(actor);
      }
    } while(r != null);
  });
}

private Runnable complete(T actor) {
  synchronized (actor) {
    Queue<Runnable> pending = pendingRunnablesOf(actor);
      if (pending.isEmpty()) {
        playingNow.remove(actor);
        return null;
      } else {
        return pending.poll();
      }
    }
  }
}
```

סעיף א:

(2)

1. סיבה: אי הוגנות - 1 נקודות
2. דוגמא נכונה המציגה מצב starvation כלשהו - 1 נקודות

המימוש החדש מנוגד לעקרון **ההוגנות**. היות ואם actor תפס thread אז הוא תופס אותו עד שמטפל בכל המשימות שלו.

דוגמא קיצונית הינה starvation אם יש אי הוגנות: actor שבאות לו משימות חדשות כל הזמן או יש לו מספר רב של משימות לטיפול - במקרה זה ה-actor לא ישחרר את ה-thread לטובת אחרים כי הוא יטפל בכולם לפני כן.

סעיף ב:

(1)

1. מי שלא הסביר מה קורה במקרה של הצלחה בשליחה של כל ה-Response - איבד 2 נקודות
2. מי שלא הסביר על מקרה של אי הצלחה של שליחת של כל ה-Response - איבד 2 נקודות
3. הסבר הכולל טענה רק שאחד ישלח, ובמלואו תמיד - איבד 3 נקודות
4. כללי מדי - איבד 3 נקודות

כשמקבלים Event מסוג Write מופעלת פונצית `handleReadWrite` שבודקת את סוגו, ואז מפעילה את continueWrite. פונקציה זו **בלולאה** שולפת את ה-Response הראשון בתור כלומר, ה-ByteBuffer במקום 0, מ-writeData. ומנסה לשלוח אותו ל-channel. אם מצליחים, אז מוחקים אותו, ובאטירציה הבאה, מבצעים את אותו תהליך (כלומר, ישלח Response השני בטור). אחרת, אם לא מצליחים לשלוח אותו, דבר שנבדק בעזרת פונקציה hasRemaining שמחזירת true אם נשאר מה לשלוח. אז continueWrite מסיימת את הרצתה. ואם ה-writeQueue מתרוקן בסוף התהליך אנו משנים את ה-interestedOps עבור ה-channel ל-OP_READ בלבד.

סעיף ב:

(2)

1. מספר מינימלי + דוגמה - 2 נקודות
2. מספר מקסימלי + דוגמה - 2 נקודות
3. מספר מקסימלי\מינימלי לא נכון, עם הסבר הגיוני - איבד 1 נקודות

מספר מינימלי של Responses: חלק מ-Response - לפחות byte אחד ישלח. בגלל שב-outBuffer של ה-channel יש byte אחד פנוי. אחר write event לא היה קורה.

מספר מקסימלי של Responses: כגודל ה-writeData של אותו actor. כלומר:

length(writeData)

וזה קורה כאשר מספר ה-bytes הפנויים ב-outBuffer של ה-channel שווה או גדול יותר מסה"כ bytes של כל ה-Responses הנמצאים ב-writeData ברגע הטיפול ב-Write Event.

סעיף ג:

(1)

1. הסבר על read ועל write כל אחד לחוד - קיבל 2 נקודות
2. הסבר על איך read ו-write משפיעות אחת על השני - 1 נקודות
3. הסבר מלא של כל המצבים שיכולים לקראת בעת נעילה כלשהי - 1 נקודות

מנעול מסוג זה מכיל 2 מנעולים, מנעול read עבור מצבי קריאה, ומנעול write עבור מצבי כתיבה כך ש-:

מנעול read - מאפשר נעילה מקבילית לא מוגבלת של threads כל עוד מנעול write איננו נעול, או אין threads הממתינים לנעול את המנעול ה-Write, אחרת ימתינו לשחרור מנעול Write ורק אז ינעלו את read.

מנעול write - מאפשר נעילת threads יחיד אם read איננו נעול, אחרת ממתין שמנעול read ישוחרר, ואז יכול לנעול. בזמן הזה על בקשה של נעילה על-read לא תתאפשר (ה-thread שביקש יכנס למצב blocking)

סעיף ג:

(2)

1. הסבר נכון - 2 נקודות
2. דוגמה נכונה - 2 נקודות

השורה הבעייתית שמצריכה שימוש במנעול זה היא:

```
actors.put(actor, pendingRunnables = new LinkedList<>());
```

שורה זו שמשנה את מבנה הנתונים יכולה לגרום לשורה הבאה:

```
Queue<Runnable> pendingRunnables = actors.get(actor);
```

להחזיר ערך לא נכון בגלל השינוי המבוצע במבנה של actors, אם גם put גם get מופעלות במקביל.

דוגמה: דבר זה יקרה כאשר T-1 של actor-1 ינסה לבצע פעולת get, ו-T-2 של actor-2 ינסה לעשות פעולת put . שינוי כזה מצריך נעילה גם עבור read וגם עבור write. היות ורוב הפעמים אנו עושים get, ולעיתים רחוקות אנו עושים put. נרצה לאפשר מקביליות מקסימלית, לכן משתמשים במנעול זה במקום synchronized למשל.

שאלה 4	(15 נקודות)

<u>סעיף א</u>:

שלושת האופציות הללו התקבלו:

Users and Properties
Properties and Rentals
Users and Rentals

שימו לב שקשר כזה מאופיין ע"י foreign key.

<u>סעיף ב</u>:

```
class _Rentals:
     def update_rec_on_guest
               (self, guest_id, property_id, check_out, rec):
     c = self._conn.cursor()
     c.execute("""
        UPDATE Rentals SET rec_on_guest = (?) WHERE
           guest_id=(?) and property_id=(?) AND check_out=(?)
     """, [rec, guest_id, property_id, check_out])
```

התקבלה גם תשובה ללא guest_id בתנאי ה-WHERE כיוון שהשדה לא באמת נחוץ להגדרת השורה.

<u>סעיף ג</u>:

```
SELECT Properties.property_id, check_in, check_out,
price_per_night FROM Properties INNER JOIN Rentals
on Properties.property_id=Rentals.property_id WHERE
host_id = $host_id AND YEAR(check_out)=$year
```