גיליון תשובות

מספר נבחן:_____

שאלה 1 (30 נקודות)

סעיף א:

```
APP **apps_repo_arr_;
```

מי שהבין שצריך מערך אך לא הבין מערך פוינטרים -3

סעיף ב:

```
MY_APPS_REPO(int n) {
                apps_repo_arr_ = new APP*[n];
for (int i = 0; i < n; i++) apps_repo_arr_[i]
                        = new PLAY_APP();
...
        }
```

מי שלא הקצה מערך של פוינטרים -2

מי שלא הקצה את האובייקטים על המערך -3

סעיף ג:

```
class MY_APPS_REPO{
...
static MY_APPS_REPO* Create(int n);
private:
        MY_APPS_REPO();
        MY_APPS_REPO(int n);
```

```
…
};
```

מי שלא זכר static הורד -3

מי שלא זכר בנאי private הורד -3

מי שהשתמש בסינגלטון ונתן תשובה שלמה - מלוא הנקודות

<u>סעיף ד</u>:

```
MY_APPS_REPO(MY_APPS_REPO&& rhs) { steal(rhs); }
void MY_APPS_REPO::steal(MY_APPS_REPO& rhs){
	apps_repo_arr_ = rhs.apps_repo_arr_;
	rhs.apps_repo_arr_ = nullptr;
}
```

מי שבsteal הקצה זכרון -4

מי שמימש דברים לא קשורים -2

שאלה 2 (30 נקודות)

מומלץ לעיין בהערות לפני שמגישים ערעור - לא ירדו סתם נקודות.

סעיף א:

התכונה הנשמרת (האינוריאנטה) צריכה לכסות את ההיבטים הבאים של עץ בינארי ממויין:

- ערכי העץ ברי השוואה, כלומר אינם null.
- לקודקוד יכולים להיות שני בנים, בן אחד, או אף בן (עלה).
- ערכו של הבן השמאלי קטן מערך הקודקוד, וערכו של הבן הימני גדול מערך הקודקוד.

יש לתאר היבטים אלו בעזרת שאילתות בסיסיות, מבלי להסתמך על מימוש כלשהוא (בלי this ושמות שדות). ההגדרה הבאה עושה זאת:

```
//@INV: getData() != null &&
        (getLeft() == null || getData().compareTo(getLeft().getData()) > 0) &&
        (getRight() == null || getData().compareTo(getRight().getData()) < 0)
```

סעיף ב:

בטיחות מוגדרת כשמירה על האינוריאנטה.

מימוש הממשק שומר על האינוריאנטה לאור שתי התובנות הבאות:

- לא ניתן למחוק קודקודים או לשנות את ערכם.
- ההוספה של קודקוד חדש היא פעולה אטומית: השמת כתובת למשתנה, כלומר ביצוע פעולת כתיבה של 64 ביט (long) לזיכרון [גם אם המעבד עובד ב 32 ביט זה מובטח, כי left ו right מוגדרים כ volatile].

לאור זאת, כאשר מעודכן הערך של left או right הערך שלו תואם את האינווריאנטה, הכתובת מעודכנת בפעולה אחת (אחרת היינו עשויים לקבל כתובת שחלקה הראשון מגיע מת'רד אחד וחלקה השני מת'רד אחר, וכתובת זו מצביעה על קודקוד אחר שערכו אינו תואם את האינווריאנטה), ולא ניתן למוחקו או לשנותו במקביל.

בתשובה נדרש לציין את הגדרת הבטיחות ואת שתי הנקודות שצוינו. אי הזכרה של האטומיות גררה הורדה של נקודה.

<u>סעיף ג</u>:

הגדרת נכונות דורשת התאמה של כל תוצאה מקבילית להרצה סדרתית של הפעולות בסדר כלשהו.
במימוש הקיים, הוספה של שני קודקודים ע"י שני ת'רדים עשויה להסתיים בהוספה של קודקוד אחד בלבד.
לדוגמא: נתון עץ עם קודקוד אחד בעל הערך 5. ת'רד אחד מבצע insert 3 ות'רד שני מבצע insert 2.
בהרצה סדרתית בסדר 3,2 נקבל עץ בו 3 הוא הבן השמאלי של 5, ו 2 הוא הבן השמאלי של 2.
בהרצה סדרתית בסדר 2,3 נקבל עץ בו 2 הוא הבן השמאלי של 5, ו 3 הוא הבן הימני של 2.
בהרצה מקבילית, לעומת זאת, ניתן לקבל עץ בו 2 הוא הבן שמאלי של 5 בלבד (כאשר הת'רד המוסיף אותו דורס את הקודקוד 3 עקב context switch לאחר בדיקת התנאי left==null), וכן מקרה הפוך בו 3 הוא הבן השמאלי של 2 בלבד.

ירדה נקודה למי שלא ציין את קריטריון הנכונות (השאלה היתה על נכונות ולא על הצבעה על 'משהו לא בסדר').
למי שציין קריטריון נכונות בו להרצה מקבילית חייבת להיות תוצאה אחת - כמו הרצה סדרתית בסדר ספציפי - ירדו 4 נקודות.

סעיף ד:

נסנכרן את המתודות של המחלקה, 'סנכרון מלא' ופשוט.
[למותר לציין, כי סנכרון מתודה אינו משנה את הממשק: הסנכרון הוא חלק מהקוד הממומש, 'סינטקטיק שוגר' ל synchronized this]

```
class SortedBinaryTreeImpl<T extends Comparable<T>>
                            implements SortedBinaryTree<T> {

      private T data;
      private SortedBinaryTree<T> left;
      private SortedBinaryTree<T> right;

      SortedBinaryTreeImpl(T data) throws Exception {
            if (data == null)
                  throw new Exception("Null value!");

            this.data = data;
            this.left = null;
            this.right = null;
      }

      public synchronized T getData() { return data; }
      public synchronized SortedBinaryTree<T> getLeft() { return left; }
      public synchronized SortedBinaryTree<T> getRight() { return right; }

      public synchronized void insert(T data) throws Exception {
            if (data == null)
                  throw new Exception("Null value!");

            if (this.data.compareTo(data) > 0) {
                  if (left == null)
                        left = new SortedBinaryTreeImpl<T>(data);
                  else
                        left.insert(data);
            } else {
                  if (right == null)
```

```
                right = new SortedBinaryTreeImpl<T>(data);
            else
                right.insert(data);
        }
    }
}
```

סעיף ה:

נשתמש במנגנון CompareAndSet, כך שהת'רד הממתין יבצע busy wait במקום לעבור למצב blocked, כפי שנלמד בכיתה:

```
public void insert(T data) throws Exception {
    if (data == null)
        throw new Exception("Null value!");

    if (this.data.compareTo(data) > 0) {
        if (!left.compareAndSet(null,new SortedBinaryTreeImpl<T>(data)))
            left.get().insert(data);
    } else {
        if (!right.compareAndSet(null,new SortedBinaryTreeImpl<T>(data)))
            right.get().insert(data);
    }
}
```

שאלה 3 (25 נקודות)

<u>סעיף א</u>:

כל תשובה שהכילה הסבר של חלקים בצורה נכונה, קיבלה ציון חלקי.
כל תשובה שהכילה הסבר נכון של כל החלקים, אבל גם הסבר שגוי, קיבלה ציון חלקי.
כל תשובה שהכילה הסבר שטחי, קיבלה ציון חלקי.

ההסבר היה צריך להכיל הסבר **מלא** של החלקים הבאים:

- מבנה הנתונים של playingNow המוודא ש-Request אחד יטופל עבור לקוח אחד בזמן נתון. אם הוא לא קיים בו, אז מכניסים אליו את ה-ConnectionHandler שלו ומטפלים ב-Request, ואם בא Request שני בזמן הטיפול ב-Request הראשון, אז ייכנס לטור המתנה עבורו. **[3 נקודות]**
- המימוש של execute מוודא שאם Request-1 התקבל לפני Request-2 אז יטופל Request-1 לפני Request-2 בגלל ששולחים את הראשון לפי ההסבר לעיל, והרצה הסדרתית של run (של-Request-1), ואז complete ש- **[3 נקודות]** בודקת אם אין עוד בתור על ידי הפעלת pendingRunnablesOf, אז מוציאה אותו מ-playingNow, אחרת מפעילים את הבא על ידי שליחת Runnable חדש עבור Request-2 להרצה. **[2 נקודות]**

<u>סעיף ב</u>:

הטור writeQueue מכיל בכל תא ByteBuffer. כל ByteBuffer הוא Response **אחד**. השאלה ביקשה לשלוח Response **אחד** ו-**במלואו**. כלומר, הפונקציה לא אמורה לצאת לפני שנשלח מלוא ה-Response.

(1)
כל פתרון ששלח Response אחד - אבל לא במלואו, קיבל חצי ניקוד.
כל פתרון ששלח את כולם, ובמלואם, קיבל חצי ניקוד.
טעות קריטית בקוד ללא קשר ללוגיקה איבד 2 נקודות.
אין איבוד נקודות בגלל סוגריים חסרים, או שימוש לא נכון ב-API.

```
public void continueWrite() {
    __if_ (!writeQueue.isEmpty()) {
      try {
        ___ByteBuffer top = writeQueue.top();______
        ___while top.hasRemaining())______________
        ______chan.write(top);____________________
        ___writeQueue.remove();___________________
        ___________________________________________
      } catch (IOException ex) {
        ex.printStackTrace();
        close();
      }
    }
    if (writeQueue.isEmpty()) {
      if (protocol.shouldTerminate()) close();
        else reactor.updateInterestedOps(chan,
                                SelectionKey.OP_READ);
    }
  }
```

(2)

כל הסבר שהכיל מידע שגוי מעורבב עם מידע נכון, איבד 1-2 נקודות לפי חומרת הטעות. כל הסבר שהכיל את המילים "הת'רד נתקע", "נכנס למצב block" או "busy-wait" (שימוש ב-**blocked** מעיד על שימוש **לא נכון** במושג אם היה שימוש בלולאה) וכדומה, קיבל 2 נקודות. מי שהסביר את הסיבה **הנכונה** שבגינה התר'ד לא מסיים מהר מספיק וגורם להאטה משמעותית של ה-Reactor קיבל עוד נקודה. הסיבה הינה בגלל שה-outBuffer של ה-chan **מלא** לכן, אי אפשר לסיים לפני שה-chan ירוקן מספיק בייטים להכיל את ה-Response שלנו במלואו.

סעיף ג:

הפתרון הכיל 4 חלקים שונים, וזה כדי **לשמור** על חלוקת משימות בשרת כמו שרשום בשאלה עצמה. יש פתרונות שעובדים אבל לא עומדים בכללים של חלוקת המשימות של ה-`Reactor` קיבלו 2 נקודות.
כל חלק שמומש בצורה נכונה קיבל 2 נקודות.
כל פתרון שפגם בגנריות של השרת, איבד 2 נקודות.
מי שלא השתמש ב-Encoder איבד 2 נקודות.
מי שלא עשה שימוש ב-Protocol איבד 2 נקודות.

הפתרון המלא והנכון צריך לממש את הסעיפים הבאים:

1. מימוש של פונקציה ב-`EchoProtocol` שמחזירה Response המכיל את הודעת "Welcome!".
2. מימוש של פונקציה ב-ConnectionHandler שמחזירת Runnable שמשתמש ב-Protocol כדי לייצר את ה-Response, להפעיל עליו את ה-Encoder, ולהוסיף אותו ל-writeData, ואז לשנות את ה-Events שהשרת מעוניין בהם ל-OP_WRITE. (נא לעיין בסעיף 4)
3. עדכון פונקצית `handleAccept` כדי להפעיל את הפונקציה של-(2) ולשלוח את ה-Runnable שנוצר ל-ThreadPool.
4. יש להפעיל את `updateInterestedOps` עם `OP_WRITE`, מי שלא עשה את זה איבד 2 נקודות, מי שעשה את זה במקום לא נכון, כלומר לפני הוספת ה-`Response` ל-`writeData`, איבד נקודה.

בדף הבא, יש מימוש מלא של ההסבר לעיל.

In MessagingProtocol: (interface)

```
public String createWelcomeResponse();
```

In EchoProtocol:

```
public String createWelcomeResponse() {
    return "Welcome!";
}
```

In NonBlockingConnectionHandler:

```
public Runnable createWelcomeTask() {
    return () -> {
        T response = protocol.createWelcomeResponse();
        writeQueue.add(ByteBuffer.wrap(
                            encdec.encode(response)));
        reactor.updateInterestedOps(chan,
                            SelectionKey.OP_READ |
                            SelectionKey.OP_WRITE);
       }
}
```

In handleAccept: (append)

```
pool.submit(handler, handler.createWelcomeTask())
```

שאלה 4 (15 נקודות)

<u>סעיף א</u>:

```
CREATE TABLE Users (
   id            INT       PRIMARY KEY,
   name          TEXT       NOT NULL
);

CREATE TABLE Properties (
   property_id          INT     PRIMARY KEY,
   host_id              INT         NOT NULL,
   description          TEXT    NOT NULL,
   price_per_night      INT         NOT NULL,

   FOREIGN KEY(host_id)     REFERENCES Users(id)
);

CREATE TABLE Rentals (
   user_id              INT      NOT NULL,
   property_id          INT     NOT NULL,
   check_in             DATE   NOT NULL,
   check_out            DATE    NOT NULL,
   rec_on_host          TEXT,
   rec_on_guest         TEXT,

   FOREIGN KEY(user_id)       REFERENCES Users(id)
   FOREIGN KEY(property_id) REFERENCES Properties(property_id)
   PRIMARY KEY(property_id,check_in)
);
```

כ-Primary Key של המחלקה Rentals התקבלו גם האופציות הבאות:

1. Property_id,check_out: המפתח הזה שקול לזה שלמעלה. אילו שבחרו שני תאריכים בעצם בחרו מפתח לא מינימלי ולכן הורדה נקודה.
2. User_id, check_in, או User_id, check_out: התשובה תחת ההנחה שאורח יכול להשכיר property אחת בלבד לכל לילה נתון. ההנחה אינה מתקיימת במציאות, אבל התקבלה.
3. User_id, property_id, check_in: תשובה זו נכונה תחת ההנחה שמספר משתמשים יכולים לחלוק דירה או חדר (property). אופציה זו לא קיימת במציאות במעמד המערכת (תמיד יש נציג אחד מול המערכת), ולא הייתה הכוונה. כמו כן, אפשר כמובן להשתמש בתאריך check_out במקום check_in.

בשאלה זו הגדרת הטבלאות אינה מונעת מידע לא תקין בתוך הDB, ובדיקת המידע המוכנס (נניח שאין חפיפת תאריכים בין הזמנות) צריכה להיעשות ברמת האפליקציה. לא למדנו בקורס להכניס תנאים לDB.

סעיף ב:

```
class User (object):
    def __init__(self, id, name):
        Self.id = id
        Self.name = name

class _Users:
    def __init__(self, conn):
        self._conn = conn

    def insert(self, user_id, name):
        self._conn.cursor().execute("""
            INSERT INTO Users (id,name) VALUES (?,?)
        """, [user_id,name])
```

סעיף ג:

```
SELECT name, check_out,rec_on_host FROM
Rentals INNER JOIN Users ON user_id = Users.id
WHERE rec_on_host NOT NULL AND Rentals.property_id=$property_id
```

התקבלו גם גרסאות טיפה שונות.

טעות נפוצה שלא הורדו עליה נקודות: צירוף הטבלה Properties בJoin נוסף. זה כמובן העמסה מיותרת על ה DB. כמו כן, טעות משמעותית במקרה זה היא תנאי Properties.host_id=Users.id. תנאי זה יוציא החוצה את כל האורחים שאינם מארחים. על תנאי כזה הורדו נקודות.

יש לשים לב שJoin מתבצע על טבלאות,ו- Select מתבצע על עמודות. יש הבדל בין תנאים שמתאימים לJoin לתנאים המתאימים לSelect.