תאריך הבחינה: 1.3.2020
שם המורה ד"ר מני אדלר
ד"ר ערן טרייסטר
ד"ר מרינה קוגן-סדצקי
פרופ' אנדרי שרף
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תש"פ
סמסטר: א
מועד: ב
משך הבחינה: שלוש שעות
חומר עזר: אין

אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

רשמו תשובותיכם בגיליון התשובות בלבד, תשובות מחוץ לגיליון לא יבדקו.

בהצלחה!

שאלה 1: זיכרון (25 נקודות)

בשאלה זו נדון ב:Smart Pointer Design Pattern, כפי שנלמד בכיתה.

א. הסטודנט אפי ניגש לממש את המחלקה Smart Pointer אך חלקים מהקוד נמחקו לו. עזרו לאפי להשלים את הקוד במקומות המקווקוים כפי שנלמד בכיתה (ניתן להוסיף מתודות עזר לפי הנדרש) [15 נקודות]

```
template <typename T> class SmartPointer {
	T *ptr_;
	 ----;


public:
	SmartPointer(T *ptr) {
		----
	}
	SmartPointer(const SmartPointer<T>& rhs) {
		----
	}
	SmartPointer<T>& operator=(const SmartPointer<T>& rhs) {
		----
	}
	virtual ~SmartPointer() {
		----
	}
```

```
};
```

ב. אפי התהדר בידיעותיו בC++ ולכן כתב את הקוד הבא. מה צריך להוסיף למחלקה SmartPointer על מנת שהקוד יעבוד נכון ויעיל? (רשמו אך ורק את התוספת הנדרשת, קוד מיותר יגרור איבוד נק') [10 נק]

```
SmartPointer<int> create() {
       SmartPointer<int> p(new int(42));
       return p;
}

void main() {
       SmartPointer<int> pointer = create();
}
```

שאלה 2: מקביליות (30 נקודות)

בשאלה זו נעסוק בסנכרון על ידי משתנים ופעולות אטומיות.

א.

א.1 [6 נק`]

נתונים שני תרדים t1 ו t2.
בפונקציות ()run של t1 וגם של t2 מופיעה קריאה לפונקציה ()f:

```
t1:
void run() {
    ...
    f();
    ...
}

t2:
void run() {
    ...
    f();
    ...
}
public static void main(String[] args) {
    Thread t1 = new Thread();
    Thread t2 = new Thread();
    t1.start();
    t2.start();
}
```

הראו כיצד ניתן לסנכרן, ע"י שימוש במשתנה אטומי, בין t1 ו t2 כך ש t1 יבצע את הקריאה ל f() ראשון, יסיים אותה, ורק לאחר מכן t2 יתחיל לבצע פונקציה f(). הסינכרון חייב להיות בתוך הקוד של run() שהתרדים מבצעים. לצורך הסנכרון **השתמשו במשתנה אטומי יחיד, ובפעולה האטומית CompareAndSet** שנלמדה בכיתה. כזכור, פעולת חומרה זו ניתנת לקריאה דרך מחלקות ה Atomic של Java, כמו AtomicBoolean, AtomicInteger, AtomicReference וכו'.

כתבו קוד מתאים, ונמקו את נכונותו בקצרה.

א.2 [10 נק`]

כעת יש לנו שלושה ת'רדים t1, t2, and t3. כל תרד מבצע בלולאה נצחית קטע קוד. כלומר, הקוד של שלושת התרדים נראה כך:

```
t1:
void run() {
   while (true) {
          A
   }
}

t2:
void run() {
   while (true) {
          B
   }
}

t3:
void run() {
   while (true) {
          C
   }
}
public static void main(String[] args) {
    Thread t1 = new Thread();
    Thread t2 = new Thread();
    Thread t3 = new Thread();
    t1.start();
    t2.start();
    t3.start();
}
```

כאשר A, B, ו C זה בלוק של שורות קוד שהם מבצעים (בלולאה, לנצח).

עליכם לסנכרן את התרדים כך שיתבצעו באופן הבא: קודם תרד t1 יבצע את קטע הקוד A מתחילתו ועד סופו, לאחר מכן תרד t2 יבצע את קטע הקוד B מספר כלשהו של פעמים**, מתחילתו ועד סופו, ורק לאחר מכן תרד t3 יבצע את קטע הקוד C מתחילתו ועד סופו. אחרי הסיבוב הזה הכל מתחיל מהתחלה: A, ואז B מספר כלשהו של פעמים, ואז C וכך הלאה עד שעוצרים את ריצת התוכנית.

דוגמא להרצה אסורה: קודם B, ואז פעמיים A, ואז שוב פעם B, ואז C.
לצורך הסנכרון **השתמשו בשני משתנים אטומיים בדיוק, ובפעולה האטומית CompareAndSet** שנלמדה בכיתה.
כתבו קוד מתאים, ונמקו את תשובתכם בקצרה.

** שימו לב: מספר הפעמים שקטע הקוד B מתבצע ברצף אינו נתון מראש, הדבר תלוי בתזמון הת'רדים בסוף ביצוע קטע הקוד B:
- אם t3 מתוזמן בדיוק כאשר t2 סיים לבצע את קטע הקוד שלו B, אז t3 רשאי וחייב להתחיל לבצע את קטע קוד C.
- אם t2 מתוזמן אחרי שסיים לבצע קטע קוד B ולפני ש t3 התחיל לבצע את קטע הקוד שלו C, אז t2 חוזר ומבצע שוב את קטע הקוד שלו B.

ב.

למדנו בכיתה כי ניתן לממש `LinkedList` לשימוש מקבילי, תוך סנכרון הת'רדים בעזרת הפעולה האטומית `CompareAndSet`. בן, סטודנט בקורס, חושב שניתן לממש את ההכנסה של חוליה (link) לראש הרשימה של ה `LinkedList` ללא שימוש ב synchronized, Semaphore, או CompareAndSet, באופן הבא:

```
 class Link<T> {
       private T data;
       private Link<T> next;

       Link(Link<T> next, T data) {
          this.next = next;
          this.data = data;
       }
 }

class LinkedList<T> {
```

```
      private Link<T> head = null;

      public void add(T data) {
         head = new Link<T>(head, data);
      }
}



public class Task implements Runnable {


   private volatile LinkedList<Integer> lst;

   static public volatile int id;
   static public volatile boolean isBusy = false;

   Task(LinkedList<Integer> lst) { this.lst = lst; }

   public void run () {
      while (true) {
           // do some work
           …
           // add new link to LinkedList
           while (true) {
                 if (!isBusy) {
                     id=Thread.getCurrentThreadId();
                     isBusy = true;
                      if (id == Thread.getCurrentThreadId()) {
                          lst.add(1);
                          isBusy = false;
                          break;
                      }
                 }
            }
            if (// some condition to check if enough work is done)
                 break;
      }
  }

  public static void main(String[] args) {
       LinkedList<Integer> lst = new LinkedList<Integer>();
      // create some random number of threads
      int n = random();
      for (int i=0; i<n; i++)
            (new Thread (new Task(lst))).start();
  }
}
```

ב.1 [7 נק`]

האם בפתרון של בן יתכן מצב שבו חוליות (links) שהת'רדים מנסים להכניס לרשימה המקושרת הולכים לאיבוד?

כלומר, תרד ביצע הכנסה לרשימה המקושרת, אך הרשימה המקושרת **אינה מכילה את החולייה שהוכנסה**.

אם לא, נמקו את תשובתכם.

אם כן, הראו תרחיש שבו זה קורה.

ב.2 [7 נק`]

האם במימוש של בן יתכן מצב של **deadlock ?**

האם במימוש של בן יתכן מצב של **livelock**?

אם לא, נמקו את תשובתכם. אם כן, הראו תרחיש שבו זה קורה.

שאלה 3: תקשורת (30 נקודות)

הממשק **Stores** מגדיר מבנה נתונים המאחסן את המחירים של מוצרים שונים בחנויות שונות:

```
public interface Stores {

   List<String> getStores();

   Map<String,Integer> getProductsOfStore(String store);

   void addProductToStore(String store, String product, int price);

}
```

המתודה **getStores** מחזירה רשימה של שמות החנויות.

המתודה **getProductsOfStore** מקבלת שם של חנות, ומחזירה את טבלת המוצרים בחנות זה - מיפוי שמות של מוצרים למחיר שלהם.

המתודה **addProductToStore** מקבלת שם של חנות, שם של מוצר, ומחיר, ומוסיפה את המוצר עם מחירו לטבלת המוצרים בחנות זו (אם המוצר כבר מופיע בחנות עם מחיר אחר, יישמר המחיר הנמוך מבין השניים).

נתון כי המחלקה **StoresImpl** מממשת באופן בטוח ונכון עבור הרצה מקבילית את הממשק **Stores**. <u>אופן מימושה אינו רלבנטי לשאלה זו</u>.

בהרצאות הראנו את תבנית ה **Command Invocation Protocol**.
כזכור, בתבנית עיצוב זו, השרת מגדיר מבנה נתונים מסוים בזיכרון ותומך בביצוע פקודות על מבנה זו. באופן זה הלקוח יכול לשלוח לשרת הודעות מסוג פקודה (אובייקט הממשמש את הממשק **Command<T>**, כאשר **T** הוא הטיפוס של מבנה הנתונים) והשרת יפעיל את הפקודה על המבנה שבזיכרון שלו (כלומר יקרא למתודה **execute** של אובייקט הפקודה, כאשר מבנה הנתונים מטיפוס **T** הוא הפרמטר).

<u>לנוחיותכם: בנספח בסוף השאלון מופיעות המחלקות של תבנית זו, כפי שנלמדו בכיתה (אין לשנות אותן, אלא רק להשתמש בהן אם נדרש)</u>

א. ממשו **<u>בגיליון התשובות</u>** את שתי הפקודות הבאות:

- **GetProductPrices**

הפקודה מקבלת שם של מוצר ומחזירה (מאובייקט ה **Stores** בשרת) את המחיר של המוצר הנתון בחנויות השונות (=טבלה הממפה שם חנות למחיר המוצר הנתון בה).

```
public class GetProductPrices implements Command<Stores> {

    ______________________________;

    public GetProductPrices (____________________) {
        ________________________________________;
    }

    @Override
    public Serializable execute(Stores stores) {
        ____________________________________________________________________
        ____________________________________________________________________
        ____________________________________________________________________
        ____________________________________________________________________
        ____________________________________________________________________
        ____________________________________________________________________
        ____________________________________________________________________
        ____________________________________________________________________
    }

}
```

- **AddStoreProducts**

הפקודה מקבלת שם של חנות ורשימת מוצרים עם מחיריהם (=טבלה הממפה שם של מוצר למחיר שלו), ומוסיפה את המוצרים ומחיריהם לחנות הנתונה (באובייקט ה **Stores** בשרת). אין ערך מוחזר.

```
public class AddStoreProducts implements Command<Stores> {

    ______________________________;
    ____________________________________________________;

    public addStoreProducts(
       ________________________,_____________________________________________) {
       ________________________________________;
       ________________________________________;
    }

    @Override
    public Serializable execute(Stores stores) {
        ____________________________________________________________________
        ____________________________________________________________________
        ____________________________________________________________________
    }
```

```
}
```

להזכירכם, כל המחלקות של **Java** מממשות **Serializable**.

[10 נקודות]

ב. השלימו **בגיליון התשובות** את התוכנית הבאה, המריצה שרת מחירים מבוסס ריאקטור בתבנית ה **Command Invocation Protocol**. השרת מקבל פקודות כמו **AddStoreProducts GetProductPrices,** ומבצע אותן על האובייקט מטיפוס **Stores** המוגדר בזיכרון של התהליך שלו.
ניתן לענות על סעיף זה גם אם לא עניתם על הסעיפים האחרים.

```
public class PriceServer {

    public static void main(String[] args) {

        Stores stores = new StoresImpl(); //one shared object

        new Reactor(
                7777, //port
                10,
                ______________________________________________, //protocol factory
                ____________________________________________//encoder-decoder factory
        ).serve();
    }
}
```

[6 נקודות]

ג. כמה ת'רדים רצים בתוכנית של סעיף ב?
ניתן לענות על סעיף זה גם אם לא עניתם על הסעיפים האחרים.

[3 נקודות]

ד. עדכנו את הקוד הנתון של סעיף ב', כך שירוץ ת'רד לכל לקוח.
ניתן לענות על סעיף זה גם אם לא עניתם על הסעיפים האחרים, ובפרט גם אם לא עניתם על סעיף ב.

[4 נקודות]

ה. ציינו תרחיש שבו שרת המחירים במודל הריאקטור יעיל יותר משרת המחירים במודל הת'רד-לכל-לקוח.
ניתן לענות על סעיף זה גם אם לא עניתם על הסעיפים האחרים.

[7 נקודות]

שאלה 4: בסיסי נתונים (15 נקודות)

בחברת Netflix משכירים סרטים ללקוחות בביתם, והם זקוקים למעקב אחר משתמשים והזמנת הסרטים שלהם לצורך המלצות. נציגי החברה פנו אליכם בבקשה ליצירת layer persistence עבורם. צריך לתמוך במידע הבא:

- ישנם משתמשים (users) אשר שוכרים סרטים (movies). לכל משתמש יש רק id ושם (name).
- לכל סרט יש id, שם הסרט (name), תיאור מילולי (description), ואופציה להיות שייך לקטגוריות (תיתכן יותר מאחת): מתח/פעולה (Action), דרמה (Drama), רומנטיקה (Romance), או ילדים (kids). כמו כן, לסרט יש דירוג: ציון ממוצע (score) וכמות מדרגים (num_scorers).
- צפייה בסרט (movie_watch) מוגדרת ע"י המשתמש הצופה בסרט, הסרט הנצפה, ותאריך הצפייה. בסיום כל צפייה הצופה יכול לדרג את הסרט (ציון מ-1 עד 10) ולהוסיף דעה (review) במלל חופשי. לשם פשטות נניח שמשתמש אינו רואה את אותו הסרט יותר מפעם אחת באותו יום.

להלן מבנה הטבלאות

```
CREATE TABLE Users (
      id     INT           PRIMARY KEY,
      name   TEXT          NOT NULL
);
CREATE TABLE Movies (
      id                   INT   PRIMARY KEY,
      name                 TEXT  NOT NULL,
      description          TEXT,
      isAction             BOOL  NOT NULL,
      isRomance            BOOL  NOT NULL,
      isDrama              BOOL  NOT NULL,
      isKids               BOOL  NOT NULL,
      score                FLOAT NOT NULL,
      num_scorers          INT   NOT NULL
);
CREATE TABLE Movie_watches (
      user_id              INT   NOT NULL,
      movie_id             INT   NOT NULL,
      date                 DATE  NOT NULL,
      score                INT,
      review               TEXT,

      FOREIGN KEY(user_id)     REFERENCES Users(id)
      FOREIGN KEY(movie_id)    REFERENCES Movies(id)
      PRIMARY KEY(user_id, movie_id, date)
);
```

סעיף א' [4 נק']

רוצים לבנות את אובייקט ה-DAO בעבור הטבלה Movies. עלינו לתמוך בפקודות הבאות:

```
UpdateScoreForMovie(movie_id, additional_score)
```

מתודה זו אחראית לעדכן score ממוצע של סרט בעקבות ציון חדש, ולהעלות באחד את השדה num_scorers. השלימו את קוד המתודה. עדכון הממוצע score מתבצע בהתאם לנוסחה:

```
new_score = (old_score*old_num_scorers + additional_score)/(old_num_scorers+1)
```

```
class _Movies:

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

def UpdateScoreForMovie(self, movie_id, new_score):
      C = self._conn.cursor()
      C.execute("""_____________DO NOT ANSWER HERE - ONLY IN ANSWER SHEET__""",
                  [_____________________________________________________])
```

להזכירכם, הקוד למטה לקוח מתוך מחלקת DAO של הטבלה Users, ומייצג עדכון שם. כמו כן, זה חוקי לעדכן ערך בטבלה בעזרת קריאת הערך הקיים ושימוש בו בחישוב, לדוגמא SET x=x+1, כאשר x שם עמודה.

```
class _Users (object)
      def update_name(self, id, name):
            self._conn.cursor().execute("""
                  UPDATE Users SET name = (?) WHERE id= (?) """, [name,id])
```

סעיף ב' [4 נק']

בהמשך לסעיף הקודם רוצים לממש את המתודה

```
GetMostPopularActionMovies(number_of_movies, minimum_selections)
```

מתודה זו אחראית לייבא את רשימת number_of_movies הסרטים הכי פופולריים מקטגוריית "פעולה" (נתרכז בקטגוריה אחת לשם פשטות). פופולריות של סרט נקבעת ע"פ הציון הממוצע של הסרט (score) בהינתן שנצפה ודורג לפחות minimum_selections פעמים. יש לייבא את את כל עמודות הטבלה.

```
def GetMostPopularActionMovies(self, number_of_movies, minimum_selections):
    self._conn.cursor().execute("""_DO NOT ANSWER HERE - ONLY IN ANSWER SHEET """,
            [_____________________________________________________])
```

כמו כן, לנוחיותכם הסינטקס של בחירה מטבלה עם תנאי בחירה, ועם מיון, כאשר ASC או DESC קובעים אם המיון בסדר עולה או יורד. כמו כן, LIMIT מגביל למספר רשומות (הראשונות בטבלה נלקחות).

```
SELECT column-list FROM table_name [WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC] [LIMIT number_records];
```

סעיף ג' [5 נק']

רוצים לדעת האם משתמש מסוים אוהב סרטים מקטגוריה מסוימת, לדוגמא סרטי פעולה. בצורה יותר ספציפית, רוצים שבהינתן צופה (user_id) נוכל להוציא (1) את כמות הסרטים שאותם ראה מקטגוריית "פעולה" (isAction=True), ו-(2) את ממוצע הציונים שנתן לסרטים שראה מקטגוריה זו. כתבו את השאילתה המתאימה. יש לוודא שבממוצע ובספירה נכנסים רק סרטים שדורגו ע"י המשתמש (בעלי ערך score מספרי).

לנוחיותכם, הסינטקס הבא סופר את כמות הרשומות ומחשב ממוצע של עמודה מסוימת בטבלה:

```
SELECT COUNT(*),AVG(Column-name) FROM Table-name
```

_____DO NOT ANSWER HERE - ONLY IN ANSWER SHEET_____

סעיף ד' [2 נק']

בהתאם ל persistence design pattern שלמדתם בקורס, האם לדעתכם השאילתה בסעיף הקודם צריכה להיות ממומשת כחלק מה DAO של אחת הטבלאות, או כחלק מה-repository? נמקו את תשובתכם.

_____DO NOT ANSWER HERE - ONLY IN ANSWER SHEET_____

<u>נספח</u>: מחלקות תבנית ה Command Invocation Protocol, כפי שהוצגו בכיתה.

```
public interface Command<T> extends Serializable {
    Serializable execute(T data);
}
```

```
public class RemoteCommandInvocationProtocol<T> implements
MessagingProtocol<Serializable> {

    private T data;

    public RemoteCommandInvocationProtocol(T data) {
        this.data = data;
    }

    @Override
    public Serializable process(Serializable msg) {
        return ((Command) msg).execute(data);
    }

    @Override
    public boolean shouldTerminate() {
        return false;
    }

}
```

```
public class ObjectEncoderDecoder implements
MessageEncoderDecoder<Serializable> {

    private final byte[] lengthBytes = new byte[4];
    private int lengthBytesIndex = 0;
    private byte[] objectBytes = null;
    private int objectBytesIndex = 0;

    @Override
    public Serializable decodeNextByte(byte nextByte) {
        if (objectBytes == null) {
           //indicates that we are still reading the length
```

```
            lengthBytes[lengthBytesIndex++] = nextByte;
            if (lengthBytesIndex == lengthBytes.length) {
                //we read 4 bytes and therefore can take the length
                int len = bytesToInt(lengthBytes);
                objectBytes = new byte[len];
                objectBytesIndex = 0;
                lengthBytesIndex = 0;
            }
        } else {
            objectBytes[objectBytesIndex++] = nextByte;
            if (objectBytesIndex == objectBytes.length) {
                Serializable result = deserializeObject();
                objectBytes = null;
                return result;
            }
        }

        return null;
    }

    private static void intToBytes(int i, byte[] b) {
        b[0] = (byte) (i >> 24);
        b[1] = (byte) (i >> 16);
        b[2] = (byte) (i >> 8);
        b[3] = (byte) i;
    }

    private static int bytesToInt(byte[] b) {
        //this is the reverse of intToBytes,
        //note that for every byte, when casting it to int,
        //it may include some changes to the sign bit
        //so we remove those by anding with 0xff

        return ((b[0] & 0xff) << 24)
                | ((b[1] & 0xff) << 16)
                | ((b[2] & 0xff) << 8)
                | (b[3] & 0xff);
    }

    @Override
    public byte[] encode(Serializable message) {
        return serializeObject(message);
    }

    private Serializable deserializeObject() {
```

```
        try {
            ObjectInput in = new ObjectInputStream(
                      new ByteArrayInputStream(objectBytes));
            return (Serializable) in.readObject();
        } catch (Exception ex) {
            throw new IllegalArgumentException(
                                "cannot desrialize object", ex);
        }

    }

    private byte[] serializeObject(Serializable message) {
        try {
            ByteArrayOutputStream bytes = new ByteArrayOutputStream();

            //placeholder for the object size
            for (int i = 0; i < 4; i++) {
                bytes.write(0);
            }

            ObjectOutput out = new ObjectOutputStream(bytes);
            out.writeObject(message);
            out.flush();
            byte[] result = bytes.toByteArray();

            //now write the object size
            intToBytes(result.length - 4, result);
            return result;

        } catch (Exception ex) {
            throw new IllegalArgumentException
                                    ("cannot serialize object", ex);
        }
    }

}
```