תאריך הבחינה: 9.2.2020
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: א
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)

רוצים לכתוב תוכנית ב C++ לאיחסון ועיבוד תמונות שחור-לבן משני סוגים שונים: תמונות דיוקן **Portrait** ותמונות נוף **Scenery**. כל פיקסל בתמונה מחזיק ערך בתחום 0-255 של דרגת אפור. על כן הוחלט לשמור תמונה במערך חד ממדי של BYTES (ע"י שרשור שורות הפיקסלים בתמונה לוקטור חד ממדי) . על מנת לאפשר שמירה של תמונות שונות יחד, הוחלט לבנות מחלקה ImageRepository אשר תאחסן תמונות שונות במערך בשם **_images.**
להלן המימוש החלקי הבא:

```
class Image {
 public:
       virtual void Filter() = 0;
 private:
       byte *_data;
};

class Portrait : public Image{
 public:
       void Filter() { ; }
};

class Scenery : public Image{
 public:
       void Filter() { ; }
};

class ImageRepository {
```

```
 public:
         ImageRepository(int capacity);
         void Insert(Image *arr, int size);
private:
       Image **_images; //מערך
        int    _capacity;//גודל מקסימלי של המערך
        int    _free; //מספר מקומות פנויים במערך
};

void main()
{
       Scenery scenes[2];
       Portrait portraits[3];

       ImageRepository myRep(5);

       myRep.Insert(scenes,2);
       myRep.Insert(portraits,3);
      /*here*/
}
```

א. ממשו את הבנאי ופונקצית Insert של המחלקה ImageRepository (מסומנים בקו). הבנאי מקצה זיכרון למערך מצביעים בגודל capacity מקומות. Insert מקבל מערך תמונות ומעתיק אותו העתקה עמוקה למערך במקום הפנוי. יש להוסיף שדות ומתודות אם נדרש. [10 נקודות]

ב. סטודנט כתב את קטע הקוד הבא:

```
ImageRepository CreateEmptyRep() {
       ImageRepository rep;
       return rep;
}

void main(){
       ImageRepository myRep(5);
       myRep = CreateEmptyRep();
}
```

הוסיפו את הנדרש (ואך ורק את הנדרש) על מנת שהקוד הנוסף יעבוד בצורה יעילה ונכונה (יורדו נקודות על קוד מיותר).
[10 נקודות]

ג. ציירו את תמונת הזיכרון המתקבלת בסוף main של סעיף א בשורה: **`/*here*/`.** יש לפרט. [5 נקודות]

שאלה 2: מקביליות (30 נקודות)

סעיף א [5 נקודות]

נתון הקוד הבא.

כתבו את כל הפלטים האפשריים של התוכנית הבאה. נמקו את תשובתכם.

```
public class A {
       public int i;
       public A(int i) { this.i = i; }
}

public class Task implements runnable {
       public A a;

       public Task(A a) { this.a = a; }

       public void run() {
              System.out.print(a.i);
       }

       public static void main(String[] args) {
              A a = new A(2);

              Thread t1 = new Thread(new Task(a));
              Thread t2 = new Thread(new Task(a));
              t1.start();
              t2.start();

              a = new A(3);
      }
}
```

סעיף ב [10 נקודות]

נתונה תוכנית המריצה שלושה ת'רדים. לרשותם של הת'רדים מאגר משאבים (resource pool) שבו נמצאים N אובייקטים (כל האובייקטים זהים אחד לשני). כאשר ת'רד מבקש אובייקט מהמאגר, ויש במאגר אובייקט פנוי (שאינו תפוס אחד הת'רדים האחרים בתוכנית), אז הת'רד תופס את

האובייקט, משתמש בו, ומשחרר אותו לאחר מכן. אם ת'רד מבקש אובייקט ואין אובייקטים פנויים כרגע במאגר, הת'רד עובר למצב BLOCKED עד שיהיה משאב פנוי במאגר.

הניחו שהמאגר הינו מבנה נתונים מסונכרן.
הניחו שבכל בקשה הת'רד תופס רק אובייקט אחד מהמאגר, אבל יכול לבצע כמה בקשות כאלו לפני שהוא מבצע את הפעולה שלו.
כאשר ת'רד מסיים את הפעולה, הוא משחרר את כל האובייקטים שהוא תפס עבורה.

ב.1 מהו המספר המינימלי של האובייקטים במאגר (כלומר, מהו הערך המינימלי של N) כך שלעולם לא יקרה מצב של חבק (deadlock), אם כל ת'רד בתוכנית צריך **2 אובייקטים** כדי לבצע את פעולתו?

ב.2 מהו המספר המינימלי של האובייקטים במאגר (כלומר, מהו הערך המינימלי של N) כך שלעולם לא יקרה מצב של חבק (deadlock) בתוכנית, אם כל ת'רד בתוכנית צריך **משאב אחד בלבד** כדי לבצע את פעולתו?

סעיף ג [15 נקודות]

בן קיבל משימה לכתוב קוד שמריץ שלושה ת'רדים, כאשר הת'רדים בוחרים 'מנהיג' אחד מביניהם.

הוא בחר בעיצוב הבא:

```
interface Leadership {
   boolean isLeader();
   void setLeader(boolean b);
}

class LeadershipBasedTask implements Runnable, Leadership {

   private boolean bLeader;
   private ConcurrentLinkedQueue<Thread> threads;

   LeadershipBasedTask(ConcurrentLinkedQueue<Thread> threads) {
       this.bLeader = true;
       this.threads = threads;
   }

   public boolean isLeader() {
       return bLeader;
   }

   public void setLeader(boolean b) {
       bLeader = b;
```

```
   }

   public void run() {
      for(Thread t : threads) {
         if(System.identityHashCode(Thread.currentThread()) <
            System.identityHashCode(t)) {
               setLeader(false);
               break;
            }
         }
      }
      //do something
   }


class Test {

   public static void run3ThreadsWithLeader() {

      ConcurrentLinkedQueue<Thread> threads =
         new ConcurrentLinkedQueue<Thread>();

       for(int i = 0; i < 3; i++) {
          LeadershipBasedTask task = new LeadershipBasedTask(threads);
          Thread thread = new Thread(task);
          threads.add(thread);
           thread.start();
       }
   }

   public static void main(String[] args) {
       run3ThreadsWithLeader();
   }
}
```

ג.1 הגדירו תנאי סיום (post condition) עבור המתודה **`run3ThreadsWithLeader`**.

ג.2 האם תנאי הסיום מתקיימים במימוש הנוכחי? נמקו.
(הניחו שהפונקציה **`identityHashCode`** מחזירה ערך שונה לכל אובייקט)

ג.3 עדכנו את הקוד כך שתנאי הסיום יתקיימו (אין לשנות את שיטת בחירת המנהיג)

שאלה 3: תקשורת (30 נקודות)

בנספח (בסוף השאלון), מופיע לנוחיותכם קוד הריאקטור כפי שנלמד בכיתה.

כדי לעקוב אחר עבודת השרת, הוחלט להוסיף לתבנית הריאקטור מנגנון logging, המדווח בקובץ reactor.log על פעולות שונות שהשרת ביצע.

לשם כך הוגדרה המחלקה Logger:

```
public class Logger {

	OutputStream out;

	Logger(String filename) throws IOException {
		out = new FileOutputStream(filename);
	}

	public synchronized void log(byte[] msg) throws IOException {
		out.write(msg);
		out.flush();
	}
}
```

א. עדכנו את קוד הריאקטור כך שידווחו בעזרת ה logger הפעולות הבאות:

- התחברות לקוח
- שליחת בתים ללקוח
- קריאת בתים מלקוח
- ביצוע הודעה של לקוח

אין לכתוב מחדש קוד קיים בגיליון התשובות. יש לציין את המחלקה, המתודה, ולציין מה הקוד החדש והיכן הוא ממוקם.
לדוגמא:

מחלקה: **Reactor**
שדה חדש: **`Logger logger`**
מתודה: **בנאי**
הוספת שורה עם האתחול של השדה החדש

```
Logger = new Logger("reactor.log");
```

[12 נקודות]

ב. ציינו אילו ת'רדים עשויים להיתקע (לעבור למצב blocked) בעקבות השינויים בקוד, ומדוע (2 סיבות)?
אין לחרוג מהמקום שהוקצה לכך בגיליון התשובות.
[4 נקודות]

ג. נתונה המחלקה **FileChannel** המרחיבה את המחלקה **SelectableChannel** (בדומה ל **SocketChannel**) עבור קריאה וכתיבה לערוצי קלט ופלט מסוג קובץ.

בדומה למחלקה **SocketChannel**, המחלקה **FileChannel** כוללת את המתודות הבאות:

```
static FileChannel      open(String filename)
Opens a file-channel for the given file-name (if the file does not exist, it
will be created).

void  configureBlocking(boolean block)
Adjusts this channel's blocking mode.

long  read(ByteBuffer[] dsts)
Reads a sequence of bytes from this channel into the given buffers.

long  write(ByteBuffer[] srcs)
Writes a sequence of bytes to this channel from the given buffers.
```

עדכנו את קוד הריאקטור כך שהת'רדים המדווחים על הפעולות לקובץ הלוג (=קוראים למתודה log של ה Logger) לא יעברו בשל כך למצב blocked.

בדומה לסעיף א, אין לכתוב מחדש קוד קיים בגיליון התשובות. יש לציין את המחלקה, המתודה, ולציין מה הקוד החדש והיכן הוא ממוקם.

[14 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.

<u>סעיף א' (3 נק'):</u>

סטודנט בוגר הקורס כתב את השאילות הבאות ליצירת הטבלאות במסד הנתונים, אבל שכח את כל נושא המפתחות. השלימו את השאילתות כך שמסד הנתונים יוגדר בצורה יעילה. מלאו את הקוד בדף התשובות.

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

סעיף ב' (6 נק'):

ברגע שצופה בוחר סרט ומתחיל בצפייה, נוספת רשומה לטבלה movie_watches. בסוף הצפייה יש לצופה את האופציה לתת score לסרט, ולעדכן את הreview הכתוב. השלימו את מחלקות ה-DTO וה-DAO הבאות. ניתן להניח שscore הוא ציון חוקי.

```
class Movie_watch (object):
      def __init__(self, ?? Fill Answer Sheet ??):
            ?? Fill Answer Sheet ??
            ?? Fill Answer Sheet ??
            ?? Fill Answer Sheet ??

class _ Movie_watches:

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

def insert(self, user_id, movie_id, date):
      self._conn.cursor().execute("""
      ?????????????? Fill Answer Sheet ??????????????
      ?????????????? Fill Answer Sheet ??????????????? """,
      [???????????? Fill Answer Sheet ????????????????])

def update(self, user_id, movie_id, date, score, review):
      self._conn.cursor().execute("""
      ?????????????? Fill Answer Sheet ????????????????
      ?????????????? Fill Answer Sheet ????????????????? """,
      [???????????? Fill Answer Sheet ????????????????])
```

לנוחיותכם, הקוד הבא מתוך מחלקת DAO של הטבלה Users מייצג עדכון שם:

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

סעיף ג' (6 נק'):

רוצים כעת שאילתה שבהינתן מספר id של משתמש תוציא את רשימת הצפיות שלו בשנה מסוימת, הכוללת: תאריך, שם הסרט, ה-score וה-review עבור כל צפייה. רשמו את השאילתה המתאימה בשפת SQL כאשר ניתן להתייחס לשנה כמשתנה $year, ולמשתנה ה-id בתור $id. רשמו את השאילתה בדף התשובות.

לנוחיותכם, בדוגמא הבאה "שולפים" שנה מתוך תאריך ומשווים את הערך שלה

```
SELECT * FROM my_friends WHERE YEAR(birthday) = 2013
```

נספח: קוד הריאקטור, כפי שנלמד בכיתה

```
public class Reactor<T> implements Server<T> {

    private final int port;
    private final Supplier<MessagingProtocol<T>> protocolFactory;
    private final Supplier<MessageEncoderDecoder<T>> readerFactory;
    private final ActorThreadPool<NonBlockingConnectionHandler> pool;
    private Selector selector;

    private Thread selectorThread;
    private final ConcurrentLinkedQueue<Runnable> selectorTasks = new
                                            ConcurrentLinkedQueue<>();

    public Reactor(
            int numThreads,
            int port,
            Supplier<MessagingProtocol<T>> protocolFactory,
            Supplier<MessageEncoderDecoder<T>> readerFactory) {

        this.pool = new ActorThreadPool<>(numThreads);
        this.port = port;
        this.protocolFactory = protocolFactory;
        this.readerFactory = readerFactory;
    }

    public void serve() {
        selectorThread = Thread.currentThread();

        try (   Selector selector = Selector.open();
                ServerSocketChannel serverSock =
                                    ServerSocketChannel.open()) {

            this.selector = selector; //just to be able to close

            serverSock.bind(new InetSocketAddress(port));
            serverSock.configureBlocking(false);
            serverSock.register(selector, SelectionKey.OP_ACCEPT);

            while (!Thread.currentThread().isInterrupted()) {

                selector.select();
                runSelectionThreadTasks();

                for (SelectionKey key : selector.selectedKeys()) {
```

```
                if (!key.isValid()) {
                    continue;
                } else if (key.isAcceptable()) {
                    handleAccept(serverSock, selector);
                } else {
                    handleReadWrite(key);
                }
            }

            selector.selectedKeys().clear();

        }

    } catch (ClosedSelectorException ex) {
        //do nothing - server was requested to be closed
    } catch (IOException ex) {
        //this is an error
        ex.printStackTrace();
    }

    System.out.println("server closed!!!");
    pool.shutdown();
}

void updateInterestedOps(SocketChannel chan, int ops) {
    final SelectionKey key = chan.keyFor(selector);
    if (Thread.currentThread() == selectorThread) {
        key.interestOps(ops);
    } else {
        selectorTasks.add(() -> {
            if(key.isValid())
                key.interestOps(ops);
        });
        selector.wakeup();
    }
}



private void handleAccept(ServrSocketChannel serverChan,
                          Selector selector) throws IOException {
    SocketChannel clientChan = serverChan.accept();

    clientChan.configureBlocking(false);
    final NonBlockingConnectionHandler<T> handler = new
```

```
            NonBlockingConnectionHandler<>(
               readerFactory.get(),
               protocolFactory.get(),
               clientChan,
               this);

        clientChan.register(selector, SelectionKey.OP_READ, handler);
    }

 private void handleReadWrite(SelectionKey key) {
        @SuppressWarnings("unchecked")
        NonBlockingConnectionHandler<T> handler =
                    (NonBlockingConnectionHandler<T>) key.attachment();
        if (key.isReadable()) {
            Runnable task = handler.continueRead();
            if (task != null) {
                pool.submit(handler, task);
            } else if (!key.isValid()) {
                return;
            }
        }
        if (key.isWritable()) {
            handler.continueWrite();
        }
    }

  private void runSelectionThreadTasks() {
    while (!selectorTasks.isEmpty()) {
      selectorTasks.remove().run();
    }
  }

  public void close() throws IOException {
    selector.close();
  }
}


public interface MessageEncoderDecoder<T> {
  T decodeNextByte(byte nextByte);
  byte[] encode(T message);
}

public class LineMessageEncoderDecoder implements
MessageEncoderDecoder<String> {
```

```java
    private byte[] bytes = new byte[1 << 10]; //start with 1k
    private int len = 0;

    @Override
    public String decodeNextByte(byte nextByte) {
         if (nextByte == '\n') {
            return popString();
        }

        pushByte(nextByte);
        return null; //not a line yet
    }

    @Override
    public byte[] encode(String message) {
        return (message + "\n").getBytes(); //uses utf8 by default
    }

    private void pushByte(byte nextByte) {
        if (len >= bytes.length) {
            bytes = Arrays.copyOf(bytes, len * 2);
        }

        bytes[len++] = nextByte;
    }

    private String popString() {
        String result = new String(bytes, 0, len, StandardCharsets.UTF_8);
        len = 0;
        return result;
    }
}



public class EchoProtocol implements MessagingProtocol<String> {

    private boolean shouldTerminate = false;

    @Override
    public String process(String msg) {
        shouldTerminate = "bye".equals(msg);
        System.out.println("[" + LocalDateTime.now() + "]: " + msg);
        return createEcho(msg);
    }
```

```
    private String createEcho(String message) {
        String echoPart = message.substring(
                Math.max(message.length() - 2, 0), message.length());
        return message + " .. " + echoPart + " .. " + echoPart + " ..";
    }

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

```
public class NonBlockingConnectionHandler<T> implements
                                            ConnectionHandler<T> {
  private static final int BUFFER_ALLOCATION_SIZE = 1 << 13; //8k
  private static final ConcurrentLinkedQueue<ByteBuffer> BUFFER_POOL =
                                        new ConcurrentLinkedQueue<>();

  private final MessagingProtocol<T> protocol;
  private final MessageEncoderDecoder<T> encdec;
  private final Queue<ByteBuffer> writeQueue = new
                                            ConcurrentLinkedQueue<>();
  private final SocketChannel chan;
  private final Reactor reactor;

  public NonBlockingConnectionHandler(
      MessageEncoderDecoder<T> reader,
      MessagingProtocol<T> protocol,
      SocketChannel chan,
      Reactor reactor) {
        this.chan = chan;
        this.encdec = reader;
        this.protocol = protocol;
        this.reactor = reactor;
  }

  public Runnable continueRead() {
    ByteBuffer buf = leaseBuffer();
    boolean success = false;
    try {
      success = chan.read(buf) != -1;
    } catch (IOException ex) {
```

```
    ex.printStackTrace();
  }
  if (success) {
    buf.flip();
    return () -> {
      try {
        while (buf.hasRemaining()) {
          T nextMessage = encdec.decodeNextByte(buf.get());
          if (nextMessage != null) {
            T response = protocol.process(nextMessage);
            if (response != null) {
              writeQueue.add(ByteBuffer.wrap(
                                encdec.encode(response)));
              reactor.updateInterestedOps(chan, SelectionKey.OP_READ
                                              | SelectionKey.OP_WRITE);
              }
            }
          }
        } finally {
          releaseBuffer(buf);
        }
      };
    } else {
      releaseBuffer(buf);
      close();
      return null;
    }
  }

public void close() {
  try {
    chan.close();
  } catch (IOException ex) {
      ex.printStackTrace();
  }
}

public void continueWrite() {
  while (!writeQueue.isEmpty()) {
    try {
      ByteBuffer top = writeQueue.peek();
      chan.write(top);
      if (top.hasRemaining()) {
        return;
      } else {
        writeQueue.remove();
      }
```

```
      } catch (IOException ex) {
        ex.printStackTrace();
        close();
      }
    }
    if (writeQueue.isEmpty()) {
      if (protocol.shouldTerminate()) close();
        else reactor.updateInterestedOps(chan, SelectionKey.OP_READ);
    }
  }

  private static ByteBuffer leaseBuffer() {
    ByteBuffer buff = BUFFER_POOL.poll();
    if (buff == null) {
      return ByteBuffer.allocateDirect(BUFFER_ALLOCATION_SIZE);
    }
    buff.clear();
    return buff;
  }

  private static void releaseBuffer(ByteBuffer buff) {
    BUFFER_POOL.add(buff);
  }
}



public class ActorThreadPool<T> {

  private final Map<T, Queue<Runnable>> actors;
  private final ReadWriteLock actorsReadWriteLock;
  private final Set<T> playingNow;
  private final ExecutorService threads;

  public ActorThreadPool(int threads) {
      this.threads = Executors.newFixedThreadPool(threads);
      actors = new WeakHashMap<>();
      playingNow = ConcurrentHashMap.newKeySet();
      actorsReadWriteLock = new ReentrantReadWriteLock();
  }

  public void shutdown() {
    threads.shutdownNow();
  }

  public void submit(T actor, Runnable r) {
    synchronized (actor) {
```

```
      if (!playingNow.contains(actor)) {
        playingNow.add(actor);
        execute(r, actor);
      } else {
        pendingRunnablesOf(actor).add(r);
      }
    }
  }

  private Queue<Runnable> pendingRunnablesOf(T actors) {
    actorsReadWriteLock.readLock().lock();
    Queue<Runnable> pendingRunnables = actors.get(actors);
    actorsReadWriteLock.readLock().unlock();

    if (pendingRunnables == null) {
      actorsReadWriteLock.writeLock().lock();
      actors.put(actor, pendingRunnables = new LinkedList<>());
      actorsReadWriteLock.writeLock().unlock();
    }
    return pendingRunnables;
  }

  private void execute(Runnable r, T actor) {
    threads.submit(() -> {
      try {
        r.run();
      } finally {
        complete(actor);
      }
    });
  }

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