אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

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

תאריך הבחינה: 28.01.2019
שם המורה: פרופ' אנדרי שרף
ד"ר מני אדלר
ד"ר ערן טרייסטר
מר מג'יד קסיס
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב,
הנדסת תוכנה
שנה: תשע"ט
סמסטר: א
מועד: א
משך הבחינה: שלוש שעות
חומר עזר: אין

בהצלחה!

שאלה 1 (30 נקודות)

רוצים לכתוב תוכנית C++ המאפשרת אחסון של אפליקציות שונות (מסוגים שונים) יחד. נתון המימוש החלקי הבא:

```
class APP {
    virtual void run() = 0;
};

class PAY_APP : public APP{
    virtual void run() { ; }
};

class PLAY_APP : public APP{
    virtual void run() { ; }
};

class MY_APPS_REPO{

public:
```

```
    MY_APPS_REPO Clone();
    void test_apps_repo(MY_APPS_REPO repo);

private:
    ???     apps_repo_arr_;
    int     apps_repo_size_;
};
```

א. הגדירו את המערך _apps_repo_arr כך שניתן יהיה להחזיק בו מופעים משני הסוגים PLAY_APP ו PAY_APP. יש להגדיר עבור _apps_repo_arr רק טיפוס. **[5 נקודות]**

ב. ממשו בנאי המייצר ומאחסן אפליקציות מסוגים שונים לתוך _apps_repo_arr בעל החתימה הבאה:
[5 נקודות]

```
MY_APPS_REPO(int n_pay_app, int n_play_app);
```

ג.קריאה ל-new ולבנאי המחלקה הן פעולות רגישות. למדנו כי ניתן למנוע אפשרות לקרוא לבנאי של מחלקה מבחוץ על ידי הסתרתו מהמשתמש. ממשו תבנית זו עבור המחלקה MY_APPS_REPO והראו כיצד יוצרים מופעים של MY_APPS_REPO בדרך זו? **[10 נקודות]**

ד. ממשו את הנדרש על מנת שהשורת קוד הבאה תתבצע ביעילות. הניחו כי Clone ו test_apps_repo ממומשות וכמו כן repo מצביע לאובייקט תקני. יש לממש אך ורק את הפונקציות הרלוונטיות לביצוע השורה (מימוש חלקים לא נדרשים יוריד ניקוד). **[10 נקודות]**

```
repo->test_apps_repo(test->Clone());
```

שאלה 2 (30 נקודות)

נתון הממשק SortedBinaryTree המייצג עץ בינרי שערכיו ממוינים:

```
interface SortedBinaryTree<T extends Comparable<T>> {
      T getData();
      SortedBinaryTree<T> getLeft();
      SortedBinaryTree<T> getRight();
      void insert(T data) throws Exception;
}
```

א. הגדירו את התכונה הנשמרת (אינווריאנטה) של הממשק **[5 נקודות]**

נתון המימוש הבא של הממשק SortedBinaryTree:

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

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

      public 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);
            }
      }
}
```

נתונים שני ת'רדים T1,T2 עם אובייקט משותף: **`SortedBinaryTreeImpl<Integer> tree`**

ב. הסבירו בקצרה מדוע <u>הבטיחות</u> נשמרת. **[5 נקודות]**

ג. הראו תרחיש שבו לא מתקיימת <u>נכונות</u>. **[5 נקודות]**

ד. פתרו את בעיות הנכונות ע"י סנכרון התר'דים, כך שהת'רד הממתין עובר למצב `blocked`. אין לשנות את הממשק. **[5 נקודות]**

ה. פתרו את בעיות הנכונות ע"י סנכרון התר'דים, כך שהת'רד הממתין <u>אינו</u> עובר למצב `blocked` (`lock-free`). אין לשנות את הממשק.

<u>תזכורת</u>: למחלקות ה Atomic ב Java יש מתודה cas המקבלת ערך ישן וערך חדש. לדוגמא:

```
AtomicInteger counter = new AtomicInteger(0);
int oldVal = 0, newVal = 7;
boolean b = counter.compareAndSet(oldVal, newVal);
```

[10 נקודות]

שאלה 3 (25 נקודות)

בנספח למבחן מופיעה תבנית קוד הריאקטור, כפי שנלמדה בכיתה.

א. המנגנון הממומש ב-ActorThreadPool בא כדי לוודא קיום שני תנאים:
1. עבור כל לקוח מחובר לשרת, יטופל בכל זמן נתון, Request אחד בלבד.
2. עבור כל לקוח מחובר לשרת, מובטח שאם Request-1 הגיע לשרת לפני Request-2, אז Response-1 (שנוצר עבור Request-1) ישלח ללקוח לפני Response-2 (שנוצר עבור Request-2).

הסבירו איך המנגנון של הניהול ב-ActorThreadPool **מוודא** קיום שני התנאים שהוזכרו לעיל, על ידי **הדגמת** תהליך הטיפול ב-Requests של הלקוחות. **[8 נקודות]**

ב. (1) שנו את המימוש של הפונקציה continueWrite שנמצאת ב-NonBlockingConnectionHandler כך שבטיפול של Event מסוג Write הפונקציה הזו תשלח Response **אחד** ו-**במלואו**. **[6 נקודות]**

(2) הסבירו איך סעיף (1) מנוגד לרעיון הבסיסי שעליו מושתת ה- Reactor Pattern שלמדנו בכיתה. ואיך שינוי זה פוגע בביצועים של השרת. **[3 נקודות]**

ג. עבור EchoProtocol, ו-LineMessageEncoderDecoder שלמדנו בכיתה, נרצה לשנות את מימוש של השרת כך שתישלח הודעת "Welcome!" ללקוח ישר אחרי שאישרנו את התחברותו (האישור ממומש ב-handleAccept). שנו את המימוש של השרת והדגישו את השינויים בקוד שלכם כדי לבצע משימה זו בהצלחה. יש לשמור על חלוקה נכונה של המשימות בשרת! **אפשר** להוסיף פונקציות עזר ולקרוא להם לפי צורך מכל מקום שתרצו. **[8 נקודות]**

<u>הערה</u>: הקוד השלם של EchoProtocol, LineMessageEncoderDecoder נמצאים בנספח.

שאלה 4 (15 נקודות)

נציגי חברת AirBNB פנו אליכם בבקשה ליצירת persistence layer עבורם. צריך לתמוך במידע הבא:

1. יש משתמשים (users) אשר חלקם רק שוכרים דירות/חדרים (properties), חלקם מארחים(hosts), וחלקם שניהם. לכל משתמש יש מספר סידורי id.
2. לכל דירה יש מספר סידורי property_id, מארח (host) יחיד, מחיר ללילה ותיאור. משתמש יכול להיות מארח (host) של מספר דירות.
3. השכרה (rental) מיוצגת ע"י אורח (guest), דירה (וה-host שלה), תאריך כניסה ויציאה.
4. בסיום כל השכרה (תאריך check_out), נכתבות המלצות. המארח ממליץ על האורח והאורח ממליץ על המארח בהקשר לדירה הרלוונטית (בעצם זו המלצה על הדירה).

סעיף א [5 נקודות]:

בוגר הקורס כתב את השאילתות הבאות ליצירת הטבלאות, אבל שכח את כל נושא המפתחות. עזרו לו להשלים את הגדרות המפתחות כך שמסד הנתונים יוגדר בצורה יעילה. **מלאו את הקוד בדף התשובות.**

```
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(?? Answer Sheet ??)  REFERENCES ?? Answer Sheet ??
);

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(?? Answer Sheet ??)     REFERENCES ?? Answer Sheet ??
    FOREIGN KEY(?? Answer Sheet ??)     REFERENCES ?? Answer Sheet ??
    PRIMARY KEY(?????????????????? Answer Sheet ?????????????)
);
```

סעיף ב' [5 נקודות]:

השלימו את מחלקות ה-DTO וה-DAO עבור הטבלה Users. **מלאו את הקוד בדף התשובות.**

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

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

    def insert(self, user_id, name):
        self._conn.cursor().execute("""
        ?????????????????? Answer Sheet ??????????????????????
        ?????????????????? Answer Sheet ??????????????????????
        """, [????????????? Answer Sheet ?????????????????])
```

סעיף ג [5 נקודות]:

רוצים כעת שאילתא שבהינתן דירה מסוימת (נתונה ע"י property_id), תייבא את כל ההמלצות של האורחים שהתארחו באותה הדירה אי פעם. יש לייבא את שם האורח (name), תאריך ההמלצה (תאריך ה-check_out), ואת ההמלצה עצמה (rec_on_host). יש לוודא שההמלצה קיימת (NOT NULL), ואם לא קיימת יש לסנן את הרשומה. רשמו את השאילתא (אין צורך בקוד פייתון).

ניקוד חלקי יינתן לאילו שיספקו את user_id (במקום name), תאריך ההמלצה, ואת ההמלצה עצמה.

<u>נספח</u>: קוד הריאקטור, כפי שנלמד בכיתה

```
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);
      }
    }
  }
}
```

```java
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(ServerSocketChannel 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();
  }
}
```

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

public class LineMessageEncoderDecoder implements
MessageEncoderDecoder<String> {

    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;
    }
}
```

```java
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;
    }
}
```