אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

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

שימו לב:

על תשובות ריקות יינתן 20% מהניקוד!

בהצלחה!

תאריך הבחינה: 13.2.2017
שם המורה: ד"ר מני אדלר
ד"ר אחיה אליסף
מר בני לוטטי
פרופ' אנדרי שרף
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשע"ז
סמסטר: א'
מועד: א'
משך הבחינה: שלש שעות
חומר עזר: אסור

שאלה 1 (30 נקודות)

בשאלה זו נידון במשחק לוח. הלוח בגודל N x N ובתחילת המשחק מפוזרים עליו אקראית גביע אחד ומספר שחקנים. לכל שחקן יש אסטרטגית משחק הקובעת לאיזה כיוון ילך בכל רגע נתון. כאשר אחד השחקנים מגיע לגביע - המשחק נגמר.

להלן מימוש חלקי של המערכת:

```
public enum Direction { UP, DOWN, RIGHT, LEFT }
public enum Result { SUCCESS, FAIL, WIN, LOST }
public interface Strategy { Direction nextMove(); }
```

```
public final class Location {
   public final int i, j;
   public Location(int i, int j) { this.i = i; this.j = j; }
   @Override
   public boolean equals(Object obj) { … }
   public Location move(Direction direction) {
      switch (direction) {
         case RIGHT:
            return new Location(i, j + 1);
         case LEFT:
            return new Location(i, j - 1);
         case DOWN:
```

```
            return new Location(i + 1, j);
        case UP:
            return new Location(i - 1, j);
        default:
            throw new IllegalArgumentException();
    }
  }
}
```

```
public final class Player implements Runnable {
  private final Strategy strategy;
  private final Board board;
  private Location location;

  public Player(Board board, Strategy strategy, Location location) {
    this.board = board;
    this.strategy = strategy;
    this.location = location;
  }
  public void run() {
    while (true) {
      Location to = location.move(strategy.nextMove());
      Result result = board.move(location, to);
      if (result == Result.SUCCESS) location = to;
      else if (result == Result.WIN || result == result.LOST) break;
    }
  }
}
```

```
public interface Board {
  /** Start the game. */
  void start();

  /** Gracefully terminate the program. */
  void stop();

  /** Move a Player from Location a to Location b.
   * Result.FAIL is returned if b is out of bounds or if there is a Player at b.
   * If the game has ended, Result.LOST is returned.
   * Otherwise the move is legit and the Player is moved to b.
   * Then, if the goblet is at b, stop() is called and Result.WIN is returned.
   * Otherwise, Result.SUCCESS is returned.
   *
   * @param a The Location to move the Player from.
   * @param b The Location to move the Player to.
   * @return The result of the call.
```

```
    *  @throws IllegalArgumentException if there is no Player at a or if a is out of bounds.
    */
   Result move(Location a, Location b);
}
```

```
public static void main(String[] args) {
 1.      Direction[] directions = Direction.values(); // Get all enum values in array.
 2.      int[] counter = {0};
 3.      Strategy[] strategies = {
 4.            /* @TODO (A) */ ,
 5.            /* @TODO (A) */
 6.      };
 7.      for (int i = 0; i < 9; i++) System.out.print(strategies[1].nextMove() + " ");
 8.      // Prints: UP DOWN RIGHT LEFT UP DOWN RIGHT LEFT UP
 9.      int size =  Integer.parseInt(args[0]);
10.      Board board = /* @TODO (B) */;
11.      board.start();
}
```

א) יש להשלים את שורות 4 ו-5 ב-main באופן הבא: (6 נקודות)

1. בשורה 4 יש להגדיר Strategy שהקריאה ל nextMove מחזירה Direction אקראי.
2. בשורה 5 יש להגדיר Strategy כך שבכל קריאה ל nextMove מוחזר האיבר הבא ב-directions בצורה מעגלית, כפי שניתן להבין משורות 7-8.
3. אין להוסיף קוד מעבר לשורות אלו.

ב) כתבו את המחלקה BoardImpl המממשת את Board באופן בטוח לכל חישוב מקבילי. השלימו את האתחול של board במתודת ה-main הנתונה (שורה 10). (14 נקודות)

הנחיות:

1. אין לשנות חתימה למתודות הקיימות.
2. ניתן להוסיף מתודות נוספות לפי הצורך.
3. לכל השדות והמתודות, יש לכתוב חתימה מלאה, כולל רמת גישה (public/private/protected).
4. פתרונות הנועלים את כל הלוח לא יתקבלו.
5. לא ניתן לשנות קוד מחוץ למחלקה BoardImpl.
6. אתם יכולים להניח שיש לכם את הפונקציה:

```
/** Generates a set of n random Locations within the bounds of size X size. */
private static Set<Location> getRandomLocations(int size, int n)
```

ג) כתבו את המחלקה ConcurrentBoardImpl המממשת את Board באופן בטוח לכל חישוב מקבילי, וללא שימוש במנעולים. (10 נקודות)

שאלה 2 (25 נקודות)

סטודנטים בקורס "סי-קלקל" נתבקשו לכתוב מערכת לניהול טסטים בשפת C++. על פי הדרישות, התכנה צריכה לדעת לנהל אוסף טסטים כללי באופן יעיל. הסטודנטים כתבו את הקוד הבא:

```
class TestRes{
private:
        char* description_;
};
class Test{
public:
        virtual  TestRes run() = 0;
};

class TestImplA : public Test{
public:
        virtual     TestRes run() { TestRes tr("OK!");  return tr; } //assume ctor exists
};

class TestSuite{
public:
        unsigned int    getSize() const { return size_; }
        void            setSize(unsigned int i) const { size_ = i; }
        Test*           getTest(unsigned int i) const { return test_arr_[i]; }
        void            setResult(unsigned int i, TestRes res) { result_arr_[i] = res; }
        void            initImplA(unsigned int n) ;
private:
        Test            **test_arr_;
        TestRes         *result_arr_;
        unsigned int     size_;             //size of test_arr_ and result_arr_
};

void run_tests(TestSuite &ts){
        for (unsigned int i = 0; i < ts.getSize(); i++)   {
                Test *test = ts.getTest(i);
                ts.setResult(i,test->run());
        }
}
int main(){
        TestSuite ts1;
        ts1.initImplA(10);
        run_tests(ts1);
        TestSuite ts2 = ts1;
        run_tests(ts1);
        return 0;
}
```

א) ממשו במחלקה TestSuite את הפונקציה void initImplA(unsigned int n). הפונקציה צריכה לייצר n אובייקטים מסוג TestImplA ומסוג TestRes ולשים אותם במערכים test_arr_ results_arr_ בהתאמה. (7 נקודות)

ב) הוסיפו את הנדרש למחלקה TestSuite על מנת שהשורה TestSuite ts2 = ts1 תבצע העתקה עמוקה. שימו לב, לא לכתוב מימוש מלא של כל המחלקה אלא אך ורק את הנדרש לנ"ל. (8 נקודות)

ג) לאחר בחינה מדוקדקת של עבודתם של הסטודנטים, נמצא כי השורה: ts.setResult(i,test->run()); אינה יעילה. הצע דרך על ידי שינוי המחלקה TestRes בלבד לפתרון הבעיה. (10 נקודות) רמז: rule of 5

שאלה 3 (30 נקודות)

בנספח למבחן מופיעה תבנית קוד הריאקטור, כפי שנלמדה בכיתה, וכפי שניתנה בתרגיל 3.

בקוד הנוכחי, הת'רד של ה Selector קורא ByteBuffer מה Socket של הלקוח, והת'רד ב Executor מחלץ הודעות ומבצען.
אחד הסטודנטים בקורס הציע לשנות את חלוקת העבודה, כך שהת'רד של ה Selector לא רק יקרא ByteBuffer מה Socket של הלקוח, אלא גם יחלץ את ההודעות מהבתים שנקראו ויעביר אותן, כל אחת כמשימת Runnable, לביצוע ב Executor

א) עדכנו את הקוד (בגיליון התשובות) על פי הצעתו של הסטודנט. (10 נקודות)

אחת הסטודנטיות בקורס תהתה האם במימוש הצעתו של הסטודנט ניתן לוותר על ה ActorThreadPool ולהסתפק ב Executor רגיל.
כלומר, להחליף בבנאי של המחלקה Reactor את השורה:

```
this.threads =  new ActorThreadPool(threads);
```

בשורה:

```
this.threads = Executors.newFixedThreadPool(threads);
```

ב) הראו תרחיש המדגים מדוע שינוי זה בעייתי. (10 נקודות)
[ניתן לענות על סעיף זה, גם אם לא עניתם על הסעיף הקודם]

כדי לבדוק איזו מבין שתי השיטות (זו שנלמדה בכיתה והצעתו של הסטודנט) יעילה יותר, הוחלט להוסיף למחלקת הריאקטור את המתודות ioThroughput, compThroughput:

```
/** @returns (byteRead+byteWritten)/time */
float ioThroughput () {
  //@TODO
}

/** @returns processedMessages/(time*threadPoolSize)  */
float compThroughput () {
  //@TODO
}
```

המתודה ioThroughput מחזירה את מספר הבתים שנקראו ונשלחו מכל/לכל הלקוחות, ביחס לזמן שעבר מאז החל השרת לרוץ: (byteRead+byteWritten)/time

המתודה compThroughput מחזירה את מספר ההודעות הממוצע של כל הלקוחות שעובדו ע"י פרוטוקול בכל מילי-שניה ע"י ת'רד אחד: (processedMessages/(time*threadPoolSize

כאשר:

time	הזמן שחלף מהרצת השרת עד כה (במילי-שניות)
byteRead	מספר הבתים שנקראו עד כה ע"י השרת
byteWrite	מספר הבתים שנשלחו ללקוח עד כה ע"י השרת
processedMessages	מספר ההודעות שעובדו עד כה ע"י השרת
threadPoolSize	מספר הת'רדים ב executor

ג) ממשו את המתודות ioThroughput, compThroughput. (10 נקודות)

במימוש המתודות (בגיליון התשובות) התייחסו לקוד המקורי המופיע בנספח, כך שניתן לענות על סעיף זה גם אם לא עניתם על הסעיפים הקודמים.

לנוחיותכם: המתודה System.currentTimeMillis מחזירה את הזמן הנוכחי במילי-שניות.

שאלה 4 (15 נקודות)

בשאלה זו עליכם לייצר persistence layer לחנות אינטרנטית למשקפיים. הpersistence layer יתמוך בצרכי החנות הבאים:

- הכנסת משקפיים למלאי: לכל משקפיים יש מספר-דגם ומחיר.
- רישום לקוח חדש: הלקוח ימלא את הפרטים הבאים: שם ות"ז.
- קנייה: לקוח מכניס את מספר ת"ז שלו, ומספר הדגם של המשקפיים שהוא מעוניין לקנות. החנות תנפיק ללקוח קבלה המכילה את פרטי הקנייה ומספר יחודי שמזהה אותה (מספר קבלה).
- ברור מחיר: לקוח\בעל החנות יכול לבקש להדפיס את המחיר של משקפיים מדגם מסויים.
- דוח קבלות: בעל החנות יכול לבקש להדפיס את כל הקבלות שהונפקו.

א) הסיקו אלו טבלאות צריכות להיות קיימות במסד הנתונים כדי לתמוך בדרישות החנות ובנו פונקציה בפייתון אשר מייצרת את הטבלאות במסד נתונים מסוג sqlite3. הפונקציה תקבל כפרמטר אוביקט מסוג connection ל sql-lite ובאמצעותו תייצר את כל הטבלאות הדרושות (יש להקפיד על מפתחות ראשיים וזרים). (5 נק')

הבהרות:

- בעת הקנייה ניתן להניח שהחנות היא זו שתספק את מספר הקבלה שישמר במסד הנתונים.
- ניתן להניח שמחיר של משקפיים הוא מספר שלם.

ב) רשמו בפייתון את מחלקות הDTO הדרושות לפי הטבלאות שהגדרתם. (5 נק')

ג) רשמו לכל מחלקת DTO, מחלקת DAO מתאימה (5 נק')

הבהרות:

- מחלקות הDAO יקבלו בבנאי אוביקט מסוג connection של sqllite.
- אין להשתמש במחלקת הDAO הגנרית אותה למדנו בכיתה, יש לייצר בעצמכם פונקציות כגון insert, find_all, update וכו.

- בכל מחלקת DAO, אין צורך לייצר פעולות אשר החנות, לפי הצרכים הנתונים מעלה לא דורשת (לדוגמא, עבור שום DAO אין צורך לייצר פעולות של delete מכיוון שאין אף דרישה למחוק נתונים לפי צרכי החנות, בנוסף יש מחלקות שלא דורשות find_all וכו')
- בפונקציות השונות אין צורך לדאוג לתקינות קלט או לטיפול בשגיאות

לנוחיותכם מצורפת תבנית הקוד הבאה (שמות המחלקות והפונקציות בתבנית הם לצורך הדוגמא בלבד)

```
def create_tables(conn):
   conn.execute("""
         CREATE TABLE ...
   """)

#Data Transfer Objects
class Dto1(object):
  def __init__(self, ...):

...

#Data Access Objects
class Dao1(object):
   def __init__(self, conn):
         self._conn = conn

   def insert(self, dto1):
      self._conn.execute("""
         INSERT INTO ...
      """)

   def find_all(self):
     c = self._conn.cursor()
     c.execute("""
       SELECT ... FROM ...
     """)
     return [Dto1(*row) for row in c.fetchall()]
   ...
...
```

נספח

א. תבנית הריאקטור, כפי שנלמדה בכיתה, וכפי שניתנה בתרגיל 3

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

public interface MessagingProtocol<T> {
   T process(T msg);
   boolean shouldTerminate();
}
```

```
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 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;
   }

   @Override
   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(); //clear the selected keys set so that we can know about new events
         }
      } 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(() -> {
                key.interestOps(ops);
            });
            selector.wakeup();
        }
    }

    private void handleAccept(ServerSocketChannel serverChan, Selector selector) throws IOException {
        SocketChannel clientChan = serverChan.accept();
        clientChan.configureBlocking(false);
        final NonBlockingConnectionHandler handler = new NonBlockingConnectionHandler(
                readerFactory.get(),
                protocolFactory.get(),
                clientChan,
                this);
        clientChan.register(selector, SelectionKey.OP_READ, handler);
    }

    private void handleReadWrite(SelectionKey key) {
        NonBlockingConnectionHandler handler = (NonBlockingConnectionHandler) key.attachment();
        if (key.isReadable()) {
            Runnable task = handler.continueRead();
            if (task != null) {
                pool.submit(handler, task);
            }
        }
        if (key.isValid() && key.isWritable()) {
            handler.continueWrite();
        }
    }

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

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

```
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 boolean isClosed() {
```

```
        return !chan.isOpen();
    }

    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 {
    private final Map<Object, Queue<Runnable>> acts;
    private final ReadWriteLock actsRWLock;
    private final Set<Object> playingNow;
    private final ExecutorService threads;

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

    public void submit(Object act, Runnable r) {
        synchronized (act) {
            if (!playingNow.contains(act)) {
                playingNow.add(act);
                execute(r, act);
            } else {
                pendingRunnablesOf(act).add(r);
            }
        }
    }

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

```
    }

    private Queue<Runnable> pendingRunnablesOf(Object act) {
        actsRWLock.readLock().lock();
        Queue<Runnable> pendingRunnables = acts.get(act);
        actsRWLock.readLock().unlock();
        if (pendingRunnables == null) {
            actsRWLock.writeLock().lock();
            acts.put(act, pendingRunnables = new LinkedList<>());
            actsRWLock.writeLock().unlock();
        }
        return pendingRunnables;
    }

    private void execute(Runnable r, Object act) {
        threads.execute(() -> {
            try {
                r.run();
            } finally {
                complete(act);
            }
        });
    }

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