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)

סעיף א.
התקבלו גם anonymous classes פשוטים. מעט נקודות ירדו על בעיות syntax חמורות ב lambda expression / anonymous classes (לא ירד על סינטקס של enum או random).

סעיף ב.
בעיות נפוצות שירדו עליהן נקודות:

- בריחה של this בבנאי (במקום שימוש במתודת start או factory).
- נעילת כל הלוח כך שרק שחקן אחד יכול לזוז ברגע נתון (לא משנה מה שם/כתובת האובייקט).
- נעילת שני תאים בלי resource ordering.
- נעילת a או b (הפרמטרים של move), כאשר יכולים להיות כמה מופעים שונים של אותו מיקום (משחקנים שונים).
- שימוש במשתנים נוספים (למשל shouldTerminate) בלי סנכרון / volatile.
- נעילת תא במטריצה שיכול להיות null.

בעיות נפוצות שלא ירדו עליהן נקודות:

- שימוש ב interrupt מבלי שיש thread במצב sleep/wait.
- שימוש ב join בלולאה על כל הת'רדים במקום שימוש ב executor.awate.

```java
import java.util.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import static java.util.Collections.shuffle;

public final class BoardImpl implements Board {
    private final Player board[][];
    private final Object locks[][];
    private final Player[] players;
    private final Location goblet;
    private final ExecutorService executor;
    private volatile boolean ended = false;

    private BoardImpl(int size, int numberOfPlayers, Location goblet) {
        board = new Player[size][size];
        locks = new Object[size][size];
        for (int i = 0; i < locks.length; i++) {
            for (int j = 0; j < locks.length; j++) {
                locks[i][j] = new Object();
            }
        }
        this.goblet = goblet;
        players = new Player[numberOfPlayers];
        executor = Executors.newFixedThreadPool(numberOfPlayers);
    }

    /** Generates a set of n random {@link Location}s within the bounds of size X size. */
    private static Set<Location> getRandomLocations(int size, int n) {
        List<Location> locations = new LinkedList<>();
        for (int i = 0; i < size * size; i++)
            locations.add(new Location(i / size, i % size));
        Collections.shuffle(locations);
        return new HashSet<>(locations.subList(0, n));
    }

    public static Board create(int size, Strategy[] strategies) {
        Location[] locations = getRandomLocations(size, strategies.length + 1)
                .toArray(new Location[strategies.length + 1]);
        BoardImpl board = new BoardImpl(
                size, strategies.length, locations[locations.length - 1]);
        for (int i = 0; i < locations.length - 1; i++)
            board.players[i] = new Player(board, strategies[i], locations[i]);
        return board;
    }
```

```
@Override
public void start() {
    for (Player player : players) executor.execute(player);
    synchronized (this) {
        try {
            while (!ended) {
                this.wait();
            }
        } catch (InterruptedException e) { }
    }
}

@Override
public void stop() {
    ended = true;
    executor.shutdownNow();
    synchronized (this) {
        this.notifyAll();
    }
}

private boolean checkBoundsValidity(Location l) {
    return l.i >= 0 && l.i < board.length && l.j >= 0 && l.j < board.length;
}

private Player getPlayer(Location l) {
    synchronized (getLock(l)) {
        return board[l.i][l.j];
    }
}

private void setPlayer(Location l, Player p) {
    synchronized (getLock(l)) {
        board[l.i][l.j] = p;
    }
}

private Object getLock(Location l) {
    return locks[l.i][l.j];
}

private int compareLocations(Location a, Location b) {
    if (a.i < b.i || (a.i == b.i && a.j < b.j)) return -1;
    if (a.i > b.i || (a.i == b.i && a.j > b.j)) return 1;
    return 0;
}
```

```
    @Override
    public Result move(Location a, Location b) {
        if (!checkBoundsValidity(a)) throw new IllegalArgumentException();
        if (!checkBoundsValidity(b)) return Result.FAIL;
        if (a.equals(b)) throw new IllegalArgumentException();
        if (ended) return Result.LOST;
        Object[] locks = compareLocations(a, b) < 0 ?
                new Object[]{getLock(a), getLock(b)} : new Object[]{getLock(b), getLock(a)};

        synchronized (locks[0]) {
            synchronized (locks[1]) {
                Player p = getPlayer(a);
                if (p == null) throw new IllegalArgumentException();
                if (getPlayer(b) == null) {
                    setPlayer(b, p);
                    setPlayer(a, null);
                    if (b.equals(goblet)) {
                        stop();
                        return Result.WIN;
                    } else {
                        return Result.SUCCESS;
                    }
                } else {
                    return Result.FAIL;
                }
            }
        }
    }

    public static void main(String[] args) {
        Direction[] directions = Direction.values(); // Get all enum values in an array.
        int[] counter = {0};
        Strategy[] strategies = {
                () -> directions[(int) (Math.random() * directions.length)],
                () -> directions[counter[0]++ % directions.length]
        };
        for (int i = 0; i < 9; i++) System.out.print(strategies[1].nextMove() + " ");
        // Prints: UP DOWN RIGHT LEFT UP DOWN RIGHT LEFT UP
        int size = Integer.parseInt(args[0]);
        Board b = BoardImpl.create(size, strategies);
        b.start();
    }
}
```

סעיף ג.

- מי שהשתמש במנעולים (למשל Semaphore) או שכתב כמו ב' וב-ב' השתמש במנעולים - ירד ניקוד מלא.
- מי שהחליף מבני נתונים ב concurrent וטען שזה מספיק בשביל קוד בטוח ונכון - ירדו מלוא הנקודות.
- מי שהשתמש במבנה concurrent בלי cas - ירדו רוב הנקודות.
- טעות נפוצה - שימוש ב version iterator על כל הלוח - זה מקביל לנעילת כל הלוח, מעבר לכך - זה דורש "roll back" אם הגרסה השתנה.

```
public final class AtomicBoardImpl implements Board {
   private final AtomicReference<Player> board[][];
   private final AtomicReference<Location> goblet = new AtomicReference<>();
   private final ExecutorService executor;
   private final List<Player> players = new ArrayList<>();
   private volatile boolean ended = false;

   /** creates an empty board */
   private AtomicBoardImpl(int size, int numberOfPlayers) {
      board=new AtomicReference[size][size];
      for (int i = 0; i < board.length; i++)
         for (int j = 0; j < board.length; j++)
            board[i][j]=new AtomicReference<>();
      executor = Executors.newFixedThreadPool(numberOfPlayers);
   }

   public static AtomicBoardImpl create(int size, Strategy[] strategies) {
      AtomicBoardImpl board = new AtomicBoardImpl(size, strategies.length);
      AtomicInteger i = new AtomicInteger(-1);
      // A nicer way to create random locations:
      new Random()
            .ints(0, size * size) // A stream of random numbers in the range of [0, size * size)
            .distinct() // Take only distinct numbers.
            .limit(1 + strategies.length) // Limit the stream to the size of 1 + strategies.length
                  // (for goblet + players)
            // Now, the first random we want to assign to the goblet location, and the rest for
            // the players. Since the ForEach is parallel - we used AtomicReference for the
            // goblet location.
            .forEach(rnd -> {
               Location l = new Location(rnd / size, rnd % size);
               if (!board.goblet.compareAndSet(null, l)){
                  Player p = new Player(board, strategies[i.incrementAndGet()], l);
                  board.board[l.i][l.j].set(p);
                  board.players.add(p);
               }
            });
      return board;
   }

   @Override
```

```
    public void start() { same...  }
    @Override
    public void stop() { same...  }
    private boolean checkBoundsValidity(Location l){ same…  }

    private AtomicReference<Player> get(Location l){
        return board[l.i][l.j];
    }
    @Override
    public Result move(Location a, Location b) {
        if (!checkBoundsValidity(a)) throw new IllegalArgumentException();
        if (!checkBoundsValidity(b)) return Result.FAIL;
        if (a.equals(b)) throw new IllegalArgumentException();
        if (ended) return Result.LOST;
        Player p = get(a).get();
        if (p == null) throw new IllegalArgumentException();

        if (get(b).compareAndSet(null, p)) {
            get(a).set(null); // if the cas was success, then we can remove the player from a.
            if (b.equals(goblet)) { // It is important to move even if it is the goblet, otherwise
                                    // two players can win.
                stop();
                return Result.WIN;
            } else {
                return Result.SUCCESS;
            }
        } else {
            return Result.FAIL;
        }
    }
}
```

שאלה 2 (25 נקודות)

סעיף א:

```
bool TestSuite::init(unsigned int n)
{
	test_arr_ = new Test*[n];

	for (unsigned int i = 0; i < n; i++)
		test_arr_[i] = new TestImplA();

	result_arr_ = new TestRes[n];
	size_ = n;
	return true;
}
```

סעיף ב:

```
TestSuite::TestSuite(const TestSuite& rhs)
{
	size_ = rhs.getSize();
	test_arr_ = new Test*[size_];

	for (unsigned int i = 0; i < size_; i++)
		test_arr_[i] = rhs.test_arr_[i]; test_arr_[i] = new TestImplA(*rhs.test_arr_[i])

	result_arr_ = new TestRes[size_];

	for (unsigned int i = 0; i < size_; i++)
		result_arr_[i] = rhs.result_arr_[i];
}
```

סעיף ג:

```
TestRes(TestRes && rhs) { steal(rhs); }
void TestRes::steal(TestRes &rhs)
{
	description_ = rhs.description_;
	rhs.description_ = nullptr;
}
```

שגיאות וניקוד:

א.

אי הקצאת מערכים new: הורדו 2 נק. על כל אי הקצאה. כנ"ל מי שלא ביצע הקצאת האוביקטים בtest_arr_

ב.

הורדו 2 נק. למי שלא הבין כי מדובר בבנאי מעתיק

הורדו 2-5 נק. למי שלא ביצע העתקה של כל חברי המחלקה

ג.

הורדו 2-5 נקודות למי שלא נתן הסבר מפורט ומלא של הmove copy ctor. (הכוונה למי שתיאר במילים פסאודו קוד)
הורדו 2 נקודות למי שבמקום בנאי ממש אופרטור מעתיק
הורדו 2 נקודות למי ששכח NULLPTR

שאלה 3	(30 נקודות)

סעיף א

יש לעדכן את המתודה continueRead במחלקה NonBlockingConnectionHandler, כך שחילוץ ההודעות ייעשה ע"י הת'רד הנוכחי (הת'רד של ה Selector). הערך המוחזר הוא **רשימת משימות** להוספה ל executor, משימה לכל הודעה. השינויים בקוד מסומנים במרקר:

```
public List<Runnable> continueRead() {
    ByteBuffer buf = leaseBuffer();

    boolean success = false;
    try {
        success = chan.read(buf) != -1;
    } catch (ClosedByInterruptException ex) {
        Thread.currentThread().interrupt();
    } catch (IOException ex) {
        ex.printStackTrace();
    }

    if (success) {
        buf.flip();
        final List<Runnable> ret = new LinkedList<Runnable>();
        return () -> {
        try {
            while (buf.hasRemaining()) {
                T nextMessage = encdec.decodeNextByte(buf.get());
                if (nextMessage != null) {
                    ret.add(() -> {
                        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);
        }
        return ret;
    } else {
        releaseBuffer(buf);
```

```
        close();
        return null;
    }
}
```

נעדכן בהתאם את המדוה handleReadWrite במחלקה Reactor כך שתוסיף את רשימת המשימות החוזרת ל exectutor.

```
private void handleReadWrite(SelectionKey key) {
    NonBlockingConnectionHandler handler =
                                        (NonBlockingConnectionHandler)key.attachment();
    if (key.isReadable()) {
        List<Runnable> tasks = handler.continueRead();
        if (tasks != null) {
            for (Runnable task : tasks)
                    pool.submit(handler,task);
        }
    }

    if (key.isWritable()) {
        handler.continueWrite();
    }
}
```

ניקוד: ירדו שלש נקודות לי שהעביר את חילוץ ההודעות לת'רד של ה Selector, אך לא טיפל במקרה שהצטברו כמה הודעות. מעבר לכך.

סעיף ב

ההצעה של הסטודנט מונעת אמנם את בעיית ערבוב הבתים שנקראו (כי יש רק ת'רד אחד שקורא מה Socket לרשימת הבתים ב MessageEncoderDecoder), אך עדיין קיימת בעיית השמירה על ביצוע ההודעות לפי הסדר.

תרחיש: התקבלו שתי הודעות מאותו לקוח A,B. ההודעות הוכנסו ע"פ סדר לתור המשימות ב Executor, ואף נלקחו משם על ידי שני ת'רדים על פי סדר זה. מכאן ואילך סדר ביצוע ההודעות תלוי במתזמן הת'רדים ב Executor, כך שלא מובטח שהן יבוצעו דווקא בסדר A,B.

ניקוד: תשובות שהתייחסו לבעיית הוגנות/הרעבה קיבלו מחצית מן הנקודות (אלו בעיות משניות ביחס לשמירה על הסדר, אם בכלל - בעיית הוגנות קיימת גם בשימוש ב ActtorThreadPool, כאשר יש לקוח עם הרבה הודעות ואין הודעות מלקוחות אחרים)

סעיף ג

יש להגדיר מונים לספירת פעולות ה i/o, לתחזק אותם תחת סנכרון במקומות הנכונים, ולממש על פיהם את המתודות.

הקוד המעודכן מסומן במרקר:

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

   …

   public AtomicInteger ioBytesNum = new AtomicInteger(0);
   public  AtomicInteger compMsgNum = new AtomicInteger(0);
   private long initialTime;
   private int numThreads;

  …



  public float ioThroughput () {
     return (float)ioBytesNum.get() / (System.currentTimeMillis() - initialTime);
  }

  public float compThroughput () {
     return (float)compMsgNum.get() /
                                 ((System.currentTimeMillis() - initialTime)*numThreads);
  }


  public Reactor(...)
      …
      this.numThreads = numThreads;
  }


  public void serve() {
     this.initialTime = System.currentTimeMillis();
      …
  }

…
}



class NonBlockingConnectionHandler {
  …

  publicRunnable continueRead() {
      …
      if (success) {
```

```
            buf.flip();
            reactor.ioBytesNum.addAndGet(buf.remaining());

            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.compMsgNum.incrementAndGet();
                                reactor.updateInterestedOps(
                                    chan, SelectionKey.OP_READ | SelectionKey.OP_WRITE);
                            }
                        }
                    }
                } finally {
                    releaseBuffer(buf);
                }
            };
        }
        ...
}

 public void continueWrite() {
        while (!writeQueue.isEmpty()) {
            try {
                ByteBuffer top = writeQueue.peek();
                reactor.ioBytesNum.addAndGet(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);
        }
    }
    ...
}
```

ניקוד:

הגדרת משתנים ומתודות - 2 נקודות
תחזוק שלושת המונים - 6 נקודות
סנכרון - 2 נקודות

עדכון של מונה במיקום לא נכון (כמו עדכון מספר ההודעות לא אחרי שההודעה בוצעה אלא אחרי שהיא חולצה או הוספה ל executor, או עדכון של מספר הבתים שנכתבו לאחר שהוספו לווקטור הפלט ולא אחרי שנכתבו ל socket) גרר פסילה של התשובה - כל מהות השאלה היה זיהוי של המקום הנכון בקוד. זאת לא שאלה על מונים אלא שאלה שבאה לבדוק הבנה בסיסית ביותר של מבנה הריאקטור.

במידה והיתה התייחסות להיבט כלשהו של סנכרון ניתנו כל שתי הנקודות על סנכרון (אם פספסתי מישהו, נא לערער). אי התייחסות לכך גררה הפחתה של שתי נקודות - סנכרון גישה לשדות במערכת רבת ת'רדים הינו דבר אלמנטרי, איננו יודעים איזה ת'רד יפעיל את המתודות שהוספנו.

שאלה 4	(15 נקודות)

<u>סעיף א</u>

```
def create_tables(conn):
    conn.executeScript("""
        CREATE TABLE products (
          type INT PRIMARY KEY,
          price INT);

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

        CREATE TABLE receipts (
          customer_id: INT,
          product_type: INT,
          num: PRIMARY KEY,

          FOREIGN KEY(customer_id) REFERENCES customers(id),
          FOREIGN KEY(product_type) REFERENCES products(type),
        );""")
```

<u>סעיף ב</u>

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

class Product(object):
    def __init__(self, type, price):
        self.type = type
        self.price = price

class Receipt(object):
    def __init__(self, num, customer_id, product_id):
        self.num = num
        self.customer_id = customer_id
        self.product_id = product_id
```

```
class Customers(object):
    def __init__(self, conn):
        self._conn = conn
    def insert(self, c):
        self._conn.execute("""
            INSERT INTO customers (name, id) VALUES (?,?)
        """, [c.name, c.id])

class Products(object):
    def __init__(self, conn):
        self._conn = conn
    def find(self, type):
        r = self._conn.cursor().execute("""
            SELECT type, price FROM products WHERE type = ?
            """, [type]).fetchone()
        return Product(*r)
def insert(self, p):
    self._conn.execute("""
      INSERT INTO products (type, price) VALUES (?,?)
    """, [p.type, p.price])

class Receipts(object):
    def __init__(self, conn):
        self._conn = conn
    def insert(self, r):
        self._conn.execute("""
            INSERT INTO receipts (num, customer_id, product_id)
     VALUES (?,?,?)
          """, [r.num, r.customer_id, r.product_id])
    def find_all(self):
        all = self._conn.cursor().execute("""
            SELECT num, customer_id, product_id FROM receipts
        """).fetchall()
        return [Receipt(*r) for r in all]
```