אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

רשמו תשובותיכם בגיליון התשובות בלבד
תשובות מחוץ לגיליון לא יבדקו.

בהצלחה!

תאריך הבחינה: 7.2.2011
שם המורה: פרופ' מיכאל אלחדד
ד"ר מני אדלר
ד"ר אנדרי שרף
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשע"א
סמסטר: א'
מועד: ב'
משך הבחינה: שלש שעות
חומר עזר: אסור

שאלה 1 (30 נקודות)

במועד א' התוודענו למערכת המדרגות הנעות.

המערכת, כזכור, מורכבת מהאובייקטים הפסיביים הבאים:

- גרם מדרגות (StairCase). על כל מדרגה יש מקום להולך רגל אחד.
- במפלס התחתון (בתחתית המדרגות) ובמפלס העליון (בראש המדרגות) יש מקום לכל הולכי הרגל
- קבוצה של הולכי רגל (Pedestrian), המצב הפנימי של הולך רגל כולל את הגובה שלו במרחב, ואסטרטגית התקדמות (Strategy).

הגובה של הולך הרגל, ניתן על ידי הגובה של המפלס התחתון ומספר המדרגות מהמפלס התחתון למקום בו הוא עומד. לדוגמא, אם גובה המפלס התחתון הוא 5, והולך הרגל נמצא 39 מדרגות מעל המפלס, נאמר כי גובהו הוא 44.

אסטרטגיית ההתקדמות, קובעת את התנהגות הולך הרגל כאשר הוא נמצא בגרם מדרגות.

קיימים שלשה מימושים לממשק Strategy, עם מתודת ההתקדמות next() המחזירה את הגובה החדש אליו יגיע הולך הרגל:

- Relaxed - הולך הרגל אינו עולה או יורד, אלא ממתין על המדרגה בה הוא עומד.
- HurryUp - הולך הרגל מנסה להתקדם עצמאית מעלה
- HurryDown - הולך הרגל מנסה להתקדם עצמאית מטה.

בנוסף מוגדרים במערכת שני סוגים של אובייקטים אקטיביים:

- Thread ('חוט') הרץ מעל המשימה StairCaseMovement על גרם מדרגות: הנעה כלפי מעלה של גרם המדרגות בקצב נתון.
- Thread ('חוט') הרץ מעל המשימה PedestrianMovement על הולך רגל: קידום הולך רגל במעלה או במורד גרם מדרגות בהתאם לאסטרטגית ההתקדמות שלו.

בסימולציה הנתונה, קיים גרם מדרגות אחד העולה מעלה, ושני הולכי רגל:

- הולך במפלס התחתון המנסה לעלות מעלה בנוסף להתקדמות המדרגות הנעות (אסטרטגיית HurryUp).
- הולך רגל במפלס העליון המנסה לרדת מטה נגד כיוון המדרגות הנעות (אסטרטגית HurryDown).

להלן הקוד הממשש את המערכת כפי שניתן במועד א, בתוספת מימוש המתודה advance() שניתנה בפיתרון.

```
interface Strategy {
  int next(int height);
}

class Relaxed implements Strategy {
  public  int next(int height) {
       return height;
  }

class HurryUp implements Strategy {
  public  int next(int height) {
    return height+1;
  }
}

class HurryDown implements Strategy {
  public  int next(int height) {
    return height-1;
  }
}
```

```
interface Pedestrian {
  int getHeight();
  void setHeight(int height);
  Strategy getStrategy();
  void advance(StairCase stairCase) throws InterruptedException;
}
```

```
class Passenger implements  Pedestrian  {
  private int _height;
  private final Strategy _strategy;

  Passenger (int height, Strategy strategy)  {  _height  = height;  _strategy  = strategy; }
  public synchronized Strategy getStrategy() { return _strategy; }
  public synchronized int getHeight() { return _height; }
  public synchronized void setHeight(int height) { _height  = height; }
  public synchronized void advance(StairCase staircase) throws InterruptedException {
    synchronized(staircase) {
      while (!(staircase != null &&
              staircase.getPedestrian(_strategy.next(getHeight()) - staircase. fromHeight()) == null &&
              (_height == staircase.fromHeight() || _height == staircase.toHeight() ||
               staircase.getPedestrian(_height - staircase. fromHeight()) == this)))
```

```
            staircase.wait();
          staircase.setPedestrian(null, _height - staircase.fromHeight());
         _height = _strategy.next(_height);
         staircase.setPedestrian(this, _height - staircase.fromHeight());
         staircase.notifyAll();
      }
   }
}
```

```
interface StairCase {
   int  fromHeight();
   int toHeight();
   Pedestrian getPedestrian(int i);
   void setPedestrian(Pedestrian pedestrian ,int i);
   int size();
   int capacity();
}

class StairCaseImpl implements StairCase {
   private final Pedestrian[] _pedestrians;
   private final int _fromHeight;
   private final int _toHeight;

   StairCaseImpl(int fromHeight, int  toHeight) throws Exception {
      if (toHeight  - fromHeight < 0)
         throw new Exception("Negative staircase definition!");
      _fromHeight = fromHeight;
      _toHeight = toHeight;
      _pedestrians  = new Pedestrian[_toHeight  - _fromHeight + 1] ;
   }

   public int  fromHeight () { return _fromHeight; }
   public int toHeight () { return _toHeight; }
   public int capacity() {  return _pedestrians.length;  }
   public synchronized int size() {
      int size=0;
      for (Pedestrian p : _pedestrians)
         if (p!=null)
            size++;
      return size;
  }
   public synchronized Pedestrian getPedestrian(int i)  throws ArrayIndexOutOfBoundsException  {
      return _pedestrians[i];
   }
```

```
  public synchronized void setPedestrian(Pedestrian p, int i) throws ArrayIndexOutOfBoundsException  {
      _pedestrians[i] = p;
  }
}
```

```
class StairCaseMovementTask  implements Runnable {
  private final StairCase _staircase;
  private final long _speed;
  StairCaseMovementTask(StairCase staircase , long speed) {  _staircase = staircase ; _speed = speed; }
  public void run() {
    while (true) {
      try {
        for (int i=_staircase .capacity()-1; i>0; i--) {
            _staircase.setPedestrian(_staircase.getPedestrian(i-1),i);
            Pedestrian p = _staircase.getPedestrian(i);
            if (p!=null)
              p.setHeight(p.getHeight()+1);
         }
         _staircase.setPedestrian(null,0);
         synchronized(_staircase) { _staircase.notifyAll(); }
        Thread.sleep(_speed);
      } catch (Exception e) {}
    }
  }
}

class PedestrianMovementTask implements Runnable {
  private final Pedestrian _pedestrian;
  private final StairCase _stairCase;
  private final long _speed;
  PedestrianMovementTask(Pedestrian pedestrian, long speed, StairCase stairCase) {
    _pedestrian = pedestrian; _speed = speed;  _stairCase = stairCase ; }
  public void run() {
    while (true) {
      try {
        _pedestrian.advance(_stairCase);
        Thread.sleep(_speed);
      } catch (Exception e) {
        return;
      }
    }
  }
}
```

```
class Simulation {
  public static void main(String[] args) throws Exception {
    StairCase upStairCase = new StairCaseImpl(1, 39);
    Pedestrian pedestrian1 = new Passenger(1,new HurryUp());
    Pedestrian pedestrian2 = new Passenger(39,new HurryDown());
    new Thread(new StairCaseMovementTask (upStairCase,1000)).start();
    new Thread(new PedestrianMovementTask (pedestrian1,1000, upStairCase)).start();
    new Thread(new PedestrianMovementTask (pedestrian2,1000, upStairCase)).start();
  }
}
```

א. הראו כיצד המימוש הנוכחי של המתודה **advance** במחלקה **Passenger** עשוי להביא את המערכת לחבק (**deadlock**). [5 נקודות]

ב. השתמשו במחלקה Semaphore, ועדכנו את מימוש המחלקה, כך שחבק זה ימנע. [10 נקודות]

חומר עזר: תיעוד המחלקה java.util.concurrent.Semaphore

Constructor Summary
Semaphore(int permits) Creates a `Semaphore` with the given number of permits.

Method Summary	
void	**acquire**() Acquires a permit from this semaphore, blocking until one is available, or the thread is interrupted.
void	**release**() Releases a permit, returning it to the semaphore.
boolean	**tryAcquire**() Acquires a permit from this semaphore, only if one is available at the time of invocation.

ניתן לענות על שני הסעיפים הבאים, גם אם לא פתרתם את סעיפים א' ו ב'.

ג. במועד א' נוכחנו כי המחלקה StairCaseImpl בטוחה תחת כל חישוב מקבילי, שכן היא שומרת על האינווריאנטה שהוגדרה עבורה. הראו כיצד הרצת הסימולציה (כלומר הרצה מקבילית של המשימות **StairCaseMovementTask** ו **PedestrianMovementTask**) אינה נכונה, מבחינת התאמת כל תוצאה מקבילית, לתוצאה של הרצה סדרתית בסדר פעולות כלשהו. פעולה, לצורך העניין, הינה ביצוע של מתודה [5 נקודות]

ד. עדכנו את הקוד כך שהוא ישמור על הנכונות [10 נקודות]

שאלה 2	(30 נקודות)

בדיוני חבר המנהלים של "מדרגות נעות לישראל" הוחלט לרכוש את מערכת המדרגות הנעות הנ"ל ולממשה ב C++, באופן הבא:

```
class StairCase {
public:
   virtual void setPedestrian(Pedestrian *pedestrian ,int i) = 0;
   virtual Pedestrian *getPedestrian(int i) = 0;
   virtual int  fromHeight() const= 0;
   virtual int  toHeight() const = 0;
   virtual int capacity() const = 0;
   virtual ~ StairCase();};

class StairCaseImpl : public StairCase {
private:
  Pedestrian  **_pedestrians;
  int _fromHeight;
  int _toHeight;
public:

StairCaseImpl(int fromHeight, int  toHeight): _fromHeight(fromHeight), _toHeight (toHeight) { _pedestrians  = new
Pedestrian*[_toHeight  - _fromHeight + 1] ;   }

virtual ~StairCaseImpl(){ delete [] _pedestrians;}

StairCaseImpl(const StairCaseImpl& other): _fromHeight(other._fromHeight), _toHeight(other. _toHeight),
_pedestrians(other. _pedestrians){}

int  fromHeight () const { return _fromHeight; }
int toHeight () const { return _toHeight; }
int capacity() const {  return _toHeight  - _fromHeight + 1;  }
Pedestrian *getPedestrian(int i) comst {      return _pedestrians[i];   }
void setPedestrian(Pedestrian *p, int i) {        _pedestrians[i] = p;   }
};
```

א. כעת נדרש לתמוך בשינוי הגדרת המדרגות, באופן יעיל, תוך כדי ריצה.
ממשו את המתודה החיצונית הבאה אשר משנה את גבולות המדרגות (אין לשנות את חתימת המתודה, ניתן לשנות את המחלקות StairCaseImpl, StairCaseI):

```
void resize(int  fromHeight, int  toHeight, StairCaseImpl &staircase);
```

בגרם המדרגות המעודכן, יש למקם את האנשים על המדרגות אחרי השינוי לפי מיקומם המקורי (= הגובה שלהם לא משתנה). אם מיקומם חורג מגבולות המדרגות החדשות, ניתן להניח כי הם כבר לא על המדרגות. [8 נקודות]

ב. כדי לבדוק את מנגנון ה **resize**, נכתבה התוכנית הקצרה הבאה:

```
void foo(StairCaseImpl staircase, int from, int to) {
  if (from==0){
     //*******here 1****************
     return;
  }
  resize(from, to, staircase);
  foo(staircase, from--, to++);
}
void main() {
   int fromHeight = 1, toHeight = 39;
   StairCaseImpl test1(fromHeight, toHeight);
   foo(test1, fromHeight, toHeight*2);
   StairCaseImpl* test2 = new StairCaseImpl(fromHeight, toHeight);
   foo(*test2, fromHeight,toHeight);
}
```

ציירו את תמונת הזיכרון_המתקבלת במיקום here1, עבור שתי הקריאות ל foo. [12 נקודות]

ג. הרצת התכנית הנ"ל עשויה להיקלע לשגיאת זיכרון בזמן ריצה.
זהו את הבעיה, ותקנו את המחלקה StairCaseImpl בהתאם? [6 נקודות]

ד. מתכנתת, בוגרת הקורס, עדכנה (על פי עקרונות עיצוב מונחה עצמים) את הגדרת test2 בפונקציה main() מ StairCaseImpl* ל StairCase*. שינוי זה גרם לבעיית קומפילציה. תקנו את הקוד כך שהבעיה תיפתר. [4 נקודות]

```
void main() {
   ...
   StairCase * test2 = new StairCaseImpl(fromHeight, toHeight);
   foo(*test2, fromHeight*2, toHeight);
}
```

שאלה 3 (30 נקודות)

במועד א' מימשנו מערכת מתקדמת בה מופעלת מרחוק הנעת המדרגות – כלומר ביצוע המשימה StairCaseMovementTask - מתוך תהליך אחר המפעיל את כל המדרגות הנעות של החברה, באתרים השונים.

```
public class StairCaseControl {
  public static void main(String[] args) {
    try {
        StairCase upStairCase = (StairCase)Naming.lookup("132.87.45.3:4004/StairCase1");
        new Thread(new StairCaseMovementTask (upStairCase,1000)).start();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

לשם כך הוגדר גרם המדרגות כ Remote

```
interface StairCase extends java.rmi.Remote {
  int  fromHeight() throws java.rmi.RemoteException;
  int toHeight() throws java.rmi.RemoteException;
  Pedestrian getPedestrian(int i) throws java.rmi.RemoteException;
  void setPedestrian(Pedestrian pedestrian ,int i) throws java.rmi.RemoteException;
  int size()throws java.rmi.RemoteException;
  int capacity()throws java.rmi.RemoteException;
}

class StairCaseImpl extends java.rmi.server.UnicastRemoteObject implements StairCase {
...
}
```

המחלקה Simulation מפרסמת ב rmiregistry את גרם המדרגות, אך אינה מריצה ת'רד מעל המשימה StairCaseMovementTask (משימה זו מבוצעת על ידי ת'רד בתהליך StairCaseControl לעיל).

```
class Simulation {
  public static void main(String[] args) throws Exception {
    StairCase upStairCase = new StairCaseImpl(1, 39);
    Naming.rebind("132.87.45.3:4004/StairCase1",upStairCase);
    Pedestrian pedestrian1 = new Passenger(1,new HurryUp());
    Pedestrian pedestrian2 = new Passenger(39,new HurryDown());
    new Thread(new PedestrianMovementTask (pedestrian1,1000, upStairCase)).start();
```

```
    new Thread(new PedestrianMovementTask (pedestrian2,1000, upStairCase)).start();
    System.out.println(pedestrian1.getHeight());
    System.out.println(upStairCase getPedestrian(0));
    System.out.println(upStairCase getPedestrian(1));
  }
}
```

א. בקוד של המחלקה Simulationנוספו שלוש שורות הדפסה, המדפיסות את הגובה של הולך הרגל הראשון, ואת התוכן של המדרגה הראשונה (בה הוא ממוקם בתחילת ההרצה) ואת התוכן של המדרגה מעליה.
הסבירו כיצד יתכן (גם לאחר התיקון הנדרש בסעיף ד של השאלה הראשונה) ששורת ההדפסה הראשונה תדפיס גובה 1 עבור הולך הרגל, אולם הוא לא נמצא במדרגה הראשונה (אינדקס 0 בגרם המדרגות המתחיל בגובה 1) אלא במדרגה השניה – כלומר, שורת ההדפסה השניה מדפיסה null ואילו שורת ההדפסה השלישית מדפיסה את הולך הרגל הראשון. [5 נקודות]

ב. כדי להתמודד עם הבעיה של סעיף א', הוגדר Pedestrian כ Remote

```
interface Pedestrian extends java.rmi.Remote {
  int getHeight() throws java.rmi.RemoteException ;
  void setHeight(int height) throws java.rmi.RemoteException;
  Strategy getStrategy()throws java.rmi.RemoteException;
  void advance(StairCase stairCase) throws java.rmi.RemoteException;
}

class Passenger implements  Pedestrian
   extends java.rmi.server.UnicastRemoteException  implements Pedestrian   {
...
}
```

בפיתרון של מועד א' צוין כי העלאת גרם המדרגות בשלושה שלבים דורשת **238** פעולות תקשורת (הלוך-חזור). כמה פעולות תקשורת ידרשו כעת? פרטו [5 נקודות]

ג. כדי לצמצם את פעולות התקשורת, הוחלט לאפיין מחדש את הממשק Pedestrian (ואת המימוש שלו Passenger) כ Serializable (במקום Remote) וכן את אופן השימוש בו במשימה StairCaseMovementTask.
עדכנו את הקוד בהתאם להצעה זו, וציינו האם בסימולציה הנתונה זה אכן חוסך פעולות תקשורת.
[10 נקודות]

ניתן לענות על הסעיף הבא גם אם לא עניתם על הסעיפים הקודמים

ד. נתון כי מימוש תקשורת ה Skel של StairCase עם ה Stub, מתבסס על תבנית ה Reactor.
בקוד המקורי של ה Reactor מתודת ה run() במחלקה ProtocolTask מסונכרנת.
סטודנט בקורס תכנות מערכות הציע, לשם ייעול, לקרוא תחילה, תחת סנכרון, את ההודעות שהתקבלו אל רשימה מקומית. אחר כך שחרר את הסנכרון (כדי לאפשר לת'רדים אחרים לעבוד על ה ProtocolTask), ורק אז לבצע את ההודעות אחת אחרי השניה:

```
class ProtocolTask implements Runnable {
  private final ServerProtocol _protocol;
  private final StringMessageTokenizer _tokenizer;
  private final ConnectionHandler _handler;
  /* The fifo queue, which holds data coming from the socket. Access to the queue is serialized, to ensure correct
  *processing order - even if more data is received by the reactor while previous data is still being processed.*/
  private final Vector<ByteBuffer> _buffers = new Vector<ByteBuffer>();
  …
  public synchronized void run() {
    // first, add all the bytes we have to the tokenizer
    synchronized (_buffers) {
      while(_buffers.size() > 0) {
        ByteBuffer buf = _buffers.remove(0);
        this._tokenizer.addBytes(buf);
      }
    }
    // now, go over all complete messages and add the to the local 'messages' list
    List<String> messages = new ArrayList<String>();
    synchronized (this) {
      while (_tokenizer.hasMessage())
        messages .add(_tokenizer.nextMessage());
    }
    // process all messages of the local 'messages' list
    for (String message : messages) {
       String response = this._protocol.processMessage(msg);
        if (response != null) {
          try {
            ByteBuffer bytes = _tokenizer.getBytesForMessage(response);
            this._handler.addOutData(bytes);
          } catch (CharacterCodingException e) { e.printStackTrace(); }
        }
      }
    }
  }
```

האם השינוי שהציע הסטודנט עשוי לפגום בפעילותן התקינה של המדרגות הנעות, בסימולציה הנתונה? נמקו תשובתכם. [10 נקודות]

הנחיה: הבהירו לעצמכם אלו ת'רדים בשרת עובדים במקביל על חלקים שונים של המצב הפנימי ב ProtocolTask, ומתי זה קורה.
עימדו על הסיבה לסינכרון כל המתודה **run()** על **this**, איזה עיקרון בפרוטוקול זה בא להבטיח?

למעוניינים (זה לא בהכרח נדרש), ניתן למצוא בנספח (בסוף המבחן) את החלקים הרלבנטיים של המחלקה **ConnectionHandle**
כמו כן נתון כי כל המתודות של Tokenizer מסונכרנות.

שאלה 4 (10 נקודות)

במועד א הגדרנו מודל נתונים הכולל מידע סטטיסטי על המשתמשים במדרגות. עבור כל גרם מדרגות:

- מיקום גרם המדרגות (שם הארץ, שם העיר, שם המתחם)
- אוריינטציה (עולה או יורד)
- רשימה של הימים בהם הוא הופעל
  - תאריך
  - מספר עוברים ביום זה
  - זמן ממוצע למעבר ממפלס למפלס

```
create table Location (
 LocationId int primary key,
 LocationName varchar(200),
 City varchar(200),
 Country varchar(100))
}

create table Staircase (
  StaircaseId int Primary Key,
  LocationId int Foreign Key references Location,
  Orientation int   // 1 = ascending, 2 = descending
)

create table StaircaseOperation (
  StaircaseId int Foreign Key references Staircase,
  OperationDay datetime,
  PassengersNumber int,
  AverageTripDuration int, // average duration in milliseconds
  Primary Key (StaircaseId, OperationDay)
 )
```

א. כעת נדרש להוסיף לכל גרם מדרגות צוות של עובדים. הנתון היחיד עבור עובד הם שמו (ניתן להניח כי שמו של עובד מזהה אותו באופן יחודי). הוסיפו את העובדים למודל הנתונים [5 נקודות]

ב. הגדירו שאילתת SQL המחזירה את שמות העובדים בגרם המדרגות הממוקם במתחם " Forth Rail Bridge" (בואכה סקוטלנד). [5 נקודות]

```
public class ConnectionHandler {
  protected final SocketChannel _sChannel;
  protected final ReactorData _data;
  protected final AsyncServerProtocol _protocol;
  protected final StringMessageTokenizer _tokenizer;
  protected Vector<ByteBuffer> _outData;
  protected final SelectionKey _skey;
  private ProtocolTask _task;
  …
   // Post data in the pending data queue, so that the connectionHandler will send it through the socket.
   // switchToReadWriteMode() subscribes this handler key to the OP_WRITE event
   // This event will immediately fire because the output buffer of the channel is empty.
   // It will keep firing as long as the output buffer is not filled.
   // When we are done sending pending data, we will unsubscribe from OP_WRITE.
  public synchronized void addOutData(ByteBuffer buf) {
    _outData.add(buf);
    switchToReadWriteMode();
  }

  // Reads incoming data from the client:. Reads some bytes from the SocketChannel
  // Create a protocolTask, to process this data, possibly generating an answer.
  // Inserts the Task to the ThreadPool
  public void read() {
    // Do not read if protocol has terminated. Only write of pending data is
    // allowed when the protocol asked to close the connection.
    if (_protocol.shouldClose())
      return;
    SocketAddress address = _sChannel.socket().getRemoteSocketAddress();
    logger.info("Reading from " + address);
    ByteBuffer buf = ByteBuffer.allocate(BUFFER_SIZE);
    int numBytesRead = 0;
    try {
      numBytesRead = _sChannel.read(buf);
    } catch (IOException e) {
      numBytesRead = -1;
    }
    if (numBytesRead == -1) {      // Is the channel closed?
      // No more bytes can be read from the channel
      logger.info("client on " + address + " has disconnected");
      closeConnection();
      // tell the protocol that the connection terminated.
     _protocol.connectionTerminated();
```

```
        return;
    }
    // Add the buffer to the protocol task
    buf.flip();
    _task.addBytes(buf);
    // Add the protocol task to the reactor which will parse and process the data
    // when a thread becomes available for it.
    _data.getExecutor().execute(_task);
  }

  // Attempts to send data to the client. if all the data has been succesfully sent, the ConnectionHandler will
  // automatically switch to read only mode, otherwise it will stay in its current mode (which is read / write).
  public synchronized void write() {
    if (_outData.size() == 0) { // if nothing left in the output string, go back to read mode
        switchToReadOnlyMode();
        return;
    }
    // If there is something to send - send the first byte buffer
    // We will return to this write() operation very soon because the selector will keep firing the OP_WRITE event
    // after we are done writing this buffer and check if there are more buffers to be sent.
    ByteBuffer buf = _outData.remove(0);
    if (buf.remaining() != 0) {
        _sChannel.write(buf);
        // Check if the buffer contains more data: we could not send all of the buffer in one write
        // (the output buffer of the socket got full). So we remember that there is more data to be sent.
        // We will receive a new OP_WRITE event when the output buffer of the socket
        // is not full anymore and complete the write operation then.
        if (buf.remaining() != 0)
            _outData.add(0, buf);
    }
    // Check if the protocol asked us to close this connection.
    // If it did, we remain open as long as there are pending data to be sent.
    // As soon as all the data has been sent, we can close the connection.
    if (_protocol.shouldClose()) {
        switchToWriteOnlyMode();
        if (buf.remaining() == 0) {
            logger.info("disconnecting client on " + _sChannel.socket().getRemoteSocketAddress());
            closeConnection();
        }
    }
  }
...
}
```