אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

רשום/רשמי תשובותיך בגיליון התשובות בלבד
תשובות מחוץ לגיליון לא יבדקו.

בהצלחה!

תאריך הבחינה: 18.2.2005
שם המורה: ד"ר מיכאל אלחדד
מני אדלר
שם הקורס: תכנות מערכות
מספר הקורס: 2031-1-202
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תשס"ה
סמסטר: א'
מועד: ב'
משך הבחינה: שלש שעות
חומר עזר: אסור

שאלה 1 (30 נקודות)

בשאלה זו נמשיך לעסוק במשחק המיתולוגי Fire (Game & Watch 1980). כזכור, במשחק זה נדרשים צמד כבאים אמיצים, בעזרת אלונקה, לקלוט בשלום ניצולים מבית בוער.

במימוש המשחק לתוכנית מחשב ב Java, הוחלט להגדיר ארבעה ת'רדים:

- Thrower מטיל אנשים מהבנין הבוער.
- UserInterface מזהה לחיצות על המקשים left ו right.
- Left עבור כל הקשה על הכפתור left (המזוהה ע"י הת'רד UserInterface), מזיז את האלונקה והכבאים מקום אחד שמאלה.
- Right עבור כל הקשה על הכפתור right (המזוהה ע"י הת'רד UserInterface), מזיז את האלונקה והכבאים מקום אחד ימינה.

בתהליך קיימים אך ורק ארבעה ת'רדים אלו (מופע אחד מכל סוג).

אל המחלקה Firemen התוודענו זה לא מכבר:

```
class Firemen
{
   public static final int MIN_X = 1;
   public static final int MAX_X = 10;
   private int x_;

   // @INV: MIN_X <= x_ <= MAX_X
   Firemen() { x_ = 5;}

   public synchronized void repaint()
   {
      System.out.println(x_);
   }

   public synchronized void moveLeft()
   {
      if (x_ > MIN_X)
         x_--;
      repaint();
```

```
   }

   public synchronized void moveRight()
   {
      if (x_ < MAX_X)
         x_++;
      repaint();
   }

   public synchronized int getX() { return x_; }

}
```

בשאלה זו נתוודע לת'רדים UserInterface ו Thrower ובקשר שלהם לת'רדים Left ו Right.

בכל פעם שמתבצעת הקשה על כפתור left מופעלת המתודה **onClickLeft()** של UserInterface , ובכל פעם שמתבצעת הקשה על כפתור right מופעלת המתודה **onClickRight()** על פי הסדר שהוקש (לא רלבנטי כיצד זה קורה). נתון כי שני כפתורים לא יכולים להילחץ בו זמנית.

להלן שתי גרסאות של הת'רדים **UserInterface ,Left, Rigth**:

גירסה א:

```
class ClicksList
{
   private LinkedList list_;

   // add item to the end of the list
   public synchronized void add(String s) throws InterruptedException {
      list_.add(s);
      notifyAll();
   }

   // remove the first string in the list if equals to given parameter s
   public synchronized void get(String s) throws InterruptedException {
      while (list_.size() == 0 ||
             ! list_.getFirst().equals(s))
         wait();
      list_.removeFirst(); // Remove first element of list
      notifyAll();
   }
}

class UserInterface extends Thread
{
   private ClicksList clicksList_;
   UserInterface (ClicksList clicksList) {
         clicksList_ = clicksList;
   }

   public void onClickLeft() {
      try {
         clicksList_.add("L");
      } catch (InterruptedException e) {
      }
   }
```

```
   public void onClickRight() {
      try {
         clicksList_.add("R");
      } catch (InterruptedException e) {
      }
   }

   public void run() {
       // Get events from UI and dispatch to onClickLeft/Right
   }
}

class Left extends Thread
{
   private ClicksList clicksList_;
   private Firemen firemen_;

   Left(ClicksList clicksList, Firemen firemen) {
         firemen_ = firemen; clicksList_ = clicksList;
   }

   public void run() {
      while (true) {
         try{
            clicksList_.get("L");
            firemen_.moveLeft();
         } catch (InterruptedException e) {
         }
      }
   }
}
class Right extends Thread
{
   private ClicksList clicksList_;
   private Firemen firemen_;
   Right(ClicksList clicksList, Firemen firemen) {
         firemen_ = firemen;
         clicksList_ = clicksList;
   }
   public void run() {
      while (true) {
         try{
            clicksList_.get("R");
            firemen_.moveRight();
         } catch (InterruptedException e) {
         }
      }
   }
}
```

<u>גירסה ב</u>:

```
class UserInterface extends Thread
{
   private ClicksCounter clicksCounter_;
   UserInterface (ClicksCounter  clicksCounter) {
         clicksCounter_  = clicksCounter;
   }
   public void onClickLeft() {
```

```
      try {
         clicksCounter_.incLeft();
      } catch (InterruptedException e) {
      }
   }

   public void onClickRight() {
      try {
         clicksCounter_.incRight();
      } catch (InterruptedException e) {
      }
   }
}

class ClicksCounter
{
   private int leftNum_, rightNum_;
   ClicksCounter() { leftNum_ = 0; rightNum_ = 0; }

   public synchronized void incLeft() throws InterruptedException {
      leftNum_++;
      notifyAll();
   }

   public synchronized void incRight()throws InterruptedException {
      rightNum_++;
      notifyAll();
   }

   public synchronized int resetLeft () throws InterruptedException {
      while (leftNum_ == 0)
         wait();
      int ret = leftNum_;
      leftNum_ = 0;
      return ret;

   }
   public synchronized int resetRight() throws InterruptedException {
      while (rightNum_ == 0)
         wait();
      int ret = rightNum_;
      rightNum_ = 0;
      return ret;
   }
}


class Left extends Thread
{
   private ClicksCounter clicksCounter_;
   private Firemen firemen_;

   Left(ClicksCounter clicksCounter, Firemen firemen) {
       firemen_ = firemen;
       clicksCounter_ = clicksCounter;
   }

   public void run() {
      while (true) {
         try {
            int count = clicksCounter_.resetLeft();
```

```
            for (int i = 0; i < count; i++)
               firemen_.moveLeft();
         } catch (InterruptedException e) {
         }
      }
   }
}

class Right extends Thread
{
   private ClicksCounter clicksCounter_;
   private Firemen firemen_;

   Right(ClicksCounter clicksCounter, Firemen firemen) {
       firemen_ = firemen;
       clicksCounter_ = clicksCounter;
   }

   public void run() {
      while (true) {
         try {
            int count = clicksCounter_.resetRight();
            for (int i = 0; i < count; i++)
               firemen_.moveRight();
         } catch (InterruptedException e) {
         }
      }
   }
}
```

א. עבור סדרת הלחיצות (left,right,left) ציין/ני את כל הפלטים האפשריים עבור גירסה א, ועבור גירסה ב.
הפלט מוגדר כסדרת הערכים של _x המודפסת על המסך על ידי המתודה ()repaint של Firemen.
כפי שצוין בקוד של UserInterface, אתם יכולים להניח כי סדרת הלחיצות (left,right,left) מזוהה במתודת ()run של הת'רד UserInterface וגוררת הפעלה של **`onClickLeft()`** אחר כך **`OnClickRight()`** ואחר כך **`OnClickLeft()`**.

(10 נקודות)

ב. הוסף/הוסיפי מתודה ()onStop בגירסה א של **`UserInterface`** המפסיקה את פעולתם של הת'רדים Left ו Right. יש לשנות את הקוד הקיים כך ש Left ו Right יסיימו תחילה את כל הפעולות על Firemen בהתאם ללחיצות שכבר הוקשו.
(10 נקודות)

ג. בלולאת ה ()run של הת'רד Thrower מוטל בכל פעם קרבן victim בעמודה אקראית_x, ולאחר מכן יורד הקרבן מטה בעמודה זו. במידה והגיע הקרבן לגובה הקרקע (y_=1) וקיימת שם אלונקה, ניקוד המשחק (score_) עולה ב- 1, אחרת הוא יורד ב- 1.

```
class Victim {
    private final int x_; // Position from which the victim is thrown
    private int y_;       // Vertical position of the victim in [5,1]
    // @inv: 1 <= y_ <= 5 && MIN_X <= x_ <= MAX_X
    public Victim(int x) {
        x_ = x;
        y_ = 5;
    }
    public int getY() {
```

```
        return y_;
     }
     public int getX() {
        return x_;
     }
     public void fall() {
        if (y_ > 1)
           y_--;
    }
}

class Thrower extends Thread
{
   private Victim v_;
   private Firemen f_;
   private int score_;
   public Thrower(Firemen f) {
      f_ = f;
      v_ = null;
      score_ = 0;
   }
   public synchronized int getScore() { return score_; }
   public Victim getVictim() { return v_; }
   public void run() {
     try{
      Random random = new Random();
      while (true) {
         if (v_ == null) {
            // select a random number in range [min/max]
            int initX = random.nextInt(Firemen.MAX_X - Firemen.MIN_X) +
                        Firemen.MIN_X;
            v_ = new Victim(initX);
         }
         v_.fall();
         sleep(500);
         if (v_.getY() == 1) {
            synchronized(v_) {       // ***
               synchronized(f_) {    // ***
                  if (f_.getX() == v_.getX()) {
                     score_++;
                  } else {
                     score_--;
                  }
                  v_ = null;
               }
            }
         }
      }
     } catch (InterruptedException e) {
     }
   }
}
```

יש הטוענים כי סינכרון _v ו/או _f (בשורות המסומנות ב ***) מיותרים במערכת הנתונה, מה דעתך? נמק!
(5 נקודות)

ד. מוסיפים אילוץ שלא ניתן להזיז את האלונקה למקום בו נמצא ניצול.
עדכן/ני את המתודות moveLeft() ו moveRight() ב Firemen , ואת run() ב Thrower כך שיתמכו בדרישות החדשות.

הניחו כי קיימת ב Firemen מתודה `Thrower getThrower()` המחזירה את הת'רד Thrower.
(5 נקודות)

שאלה 2 (10 נקודות)

נתון הקובץ X.h

```
// Contents of x.h
    #include "b.h"  // class B
    #include "c.h"  // class C
    #include "d.h"  // class D
    #include <iostream>
    #include <list>
    #include <string>

    class X {
    public:
        X( const C& );
        D Function1( int, char* );
        D Function1( int, C );
        B& Function2( B );
    private:
        std::string  name_;
        std::list<C> clist_;
        D d_;
    };
```

א. הסירו את כל ה #include המיותרים **(5 נקודות)**

ב. ערכו את המחלקה X כך שלא יידרש שום include בקובץ X.h. ניתן להגדיר מחלקה חדשה בקובץ אחר, המבצעת includes, אך לא לבצע include לקובץ זה מ X.h. **(5 נקודות)**

שאלה 3 (20 נקודות)

נתונות המחלקות **Person** ו **Student**

```
class Person
{
public:
      Person(std::string& name,long id) : name_(name), id_(id) {}
      ~Person() {}

private:
      std::string& name_;
      long id_;
};


class Student : public Person
{
public:
   Student(std::string& name, long id) : Person(name,id) {}
   Student(std::string& name, long id, const std::vector<int*>& courses) :
```

```
      Person(name,id), courses_(courses) {}
   virtual ~Student() {
      std::vector<int*>::iterator it;
      for (it = courses_.begin(); it != courses_.end(); ++it)
         delete (*it);
   }
   const std::vector<int*>& getCourses() const { return courses_; }

private:
      std::vector<int*> courses_;
};
```

א. זהו את הבעיה הקיימת בכל אחד משני קטעי הקוד הבאים, ותקנו את הגדרת המחלקות בהתאם.

א.1 (5 נקודות)

```
{
   std::string name("Rina");
   std::vector<int*> courses;
   courses.push_back(new int(1345));
   Person* p = new Student(name,37,courses);
   Student* s = (Student*)p;
   std::cout << (*(s->getCourses().at(0)));
   delete p;
}
```

א.2 (7 נקודות)

```
{
   std::string name("Rina");
   std::vector<int*> courses;
   courses.push_back(new int(1345));
   {
      Student s (name,37,courses);
      std::cout << ++(*(s.getCourses().at(0))) << std::endl;
   }
   std::cout << name << " " << courses.at(0) << " "  << (*courses.at(0));
}
```

ב. תקנו את הבעיה בקטע הקוד הבא – התיקון בקטע הקוד ולא בהגדרות המחלקות **(3 נקודות)**

```
{
   Student s("Rina",37);
   std::vector<int*>::const_iterator it;
   for (it = s.getCourses().begin(); it != s.getCourses().end(); ++it)
      std::cout << (*(s. getCourses().at(0)));
}
```

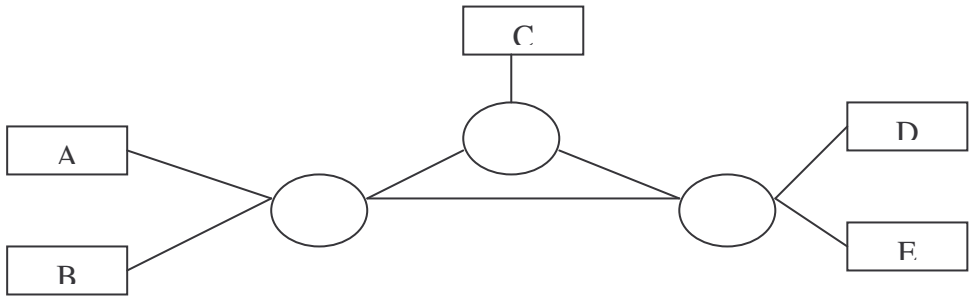
ג. נתון אובייקט מטיפוס Student בשם student.

כתבו קטע קוד, המוסיף לסטודנט את הקורס שמספרו 1200, אם לא נרשם אליו עד כה. אין לשנות את הגדרת המחלקה. **(5 נקודות)**

שאלה 4 (18 נקודות)

ציינו נכון (V) או לא נכון (X) עבור הקביעות הבאות:

א. תהליך המעונין להתחבר ל RemoteObject חייב לדעת את ה Host של התהליך בו מוגדר ה Remote Object.
ב. תקשורת בין תהליכים עם Remote Objects בטוחה יותר, מתקשורת בעזרת sockets בפרוטוקול TCP.
ג. שרת עם ת'רד אחד, הנמצא בקשר TCP עם לקוחות, יכול להשתמש ב port אחד בלבד.
ד. תהליך המאזין ל TCP socket יזהה מיד התנתקות של התהליך בצד השני.
ה. אלגוריתם Stop & Wait (Go-Back-N עבור N=1) לא חייב להשתמש במספרי הודעות.
ו. גם UDP וגם TCP משתמשים ב checksum.
ז. הודעה הנשלחת מתהליך בפרוטוקול TCP תגיע בכל מקרה לתהליך היעד.
ח. הודעה המגיעה ב multicast לתהליך, היא בהכרח בעלת תוכן שאינו שגוי.
ט. הודעה המופצת ב multicast על ידי תהליך ב Host A, לקבוצה בה חברים תהליכים ב Host B ו Host D, תבקר בלכל היותר שני routers.



שאלה 5 (12 נקודות)

נתון שרת המטפל בבקשות רבות של לקוחות רבים. לשרת משאבים גדולים והוא יכול להריץ 1000 ת'רדים.
בישיבת צוות הפיתוח הוצגו שלוש הצעות לארגון 1000 הת'רדים:

הצעה ראשונה: תהליך ובו Reactor אחד עם Thread Pool בגודל 999.

הצעה שניה: תהליך ובו מערך של עשרה Reactors, כאשר ה Thread Pool של כל Reactor הוא בגודל 99.
לקוח הרוצה להתחבר לשרת יכול לבחור, אקראית, את אחד מעשרת ה ports של כל ה Reactors המוקצים לבקשות התחברות.

הצעה שלישית: השרת יורכב מעשרה תהליכים על Host אחד. כל תהליך מורכב מ Reactor עם Thread Pool בגודל 99. לקוח הרוצה להתחבר לשרת יכול לבחור, אקראית, את אחד מעשרת ה ports של כל ה Reactors המוקצים לבקשות התחברות.

איזו הצעה תמליצו לקבל? התייחסו בתשובתכם בקצרה ליתרונות ו/או החסרונות של כל הצעה.

שאלה 6 (10 נקודות)

נתון מודל הנתונים שנבחר עבור המילון העברי:

Words

Primary Key	Str
	Freq

Analyses

Primary Key	ID
	POS
	Gender
	Number

WordsAnalyses

Primary Key	WordStr
	AnalysisId

WordStr מפתח זר לשדה Str ב Words, ו AnalysisId מפתח זר לשדה ID ב Analyses.

נניח כי לכל מילה קיימת הסתברות עבור כל אחד מניתוחיה האפשריים. לדוגמא:

עבור המילה 'מספרים' קיימים 3 ניתוחים אפשריים:

3, שם, זכר, יחיד [עבור הקריאה 'מִסְפָּרַיִם']
4, שם, זכר, רבים [עבור הקריאה 'מִסְפָּרִים']
7, פועל, זכר, רבים [עבור הקריאה 'מְסַפְּרִים']

ניתוח מספר 3 מתאים למילה 'מספרים' בהסתברות של 0.5
ניתוח מספר 4 מתאים למילה 'מספרים' בהסתברות של 0.3
ניתוח מספר 7 מתאים למילה 'מספרים' בהסתברות של 0.2

א. הוסף/הוסיפי הסתברות זו למודל הנתונים. (3 נקודות)

ב. כתוב/בי שאילתת SQL המציגה את רשימת הניתוחים – חלק דיבר (POS), מין (gender) וכמות (number) – עבור המילה 'מספרים', ממוינים בסדר עולה על פי ההסתברות שלהם. (7 נקודות)