<u>מבחן מסכם מועד ב' ב-"מבוא למדעי המחשב"</u>
<u>202-1-101-1</u>

סמסטר א' תשס"ג
18.2.2003

מר בועז בן-משה
פרופ' מיכאל קודיש
ד"ר חן קיסר
ד"ר יצחק רוזן

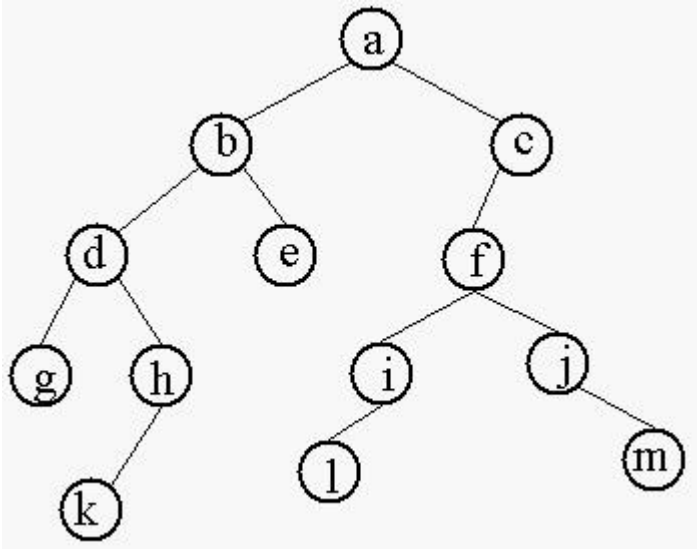
משך הבחינה שלוש שעות.
חומר עזר אסור.
אין להשתמש במחשבון.

במבחן זה 4 שאלות.

אנא רשמו את תשובותיכם בדף התשובות בלבד. המחברת שקיבלתם היא **מחברת טיוטה והיא לא תימסר כלל לבדיקה.** בסיום הבחינה נשמור אך ורק את דף התשובות. כל שאר החומר יועבר לגריסה. הקפידו לרשום בדף התשובות גם את מספר הנבחן ומספר החדר שבו אתם נבחנים.

בשאלות התכנות, מספר השורות העומדות לרשותכם בדף התשובות רומז על אורך הקוד הנדרש. הקפידו על כתב יד ברור. תשובות מסורבלות או ארוכות מדי לא יזכו בניקוד מלא. **אין צורך להעתיק את שורות הקוד הנתונות בשאלון לדף התשובות.**

בהצלחה !



<u>שאלה 1 (20 נקודות)</u>

- **תזכורת:** מסלול על עץ בין שני צמתים הוא סידרת הצלעות ("קוים" המחברים שני צמתים) שאותן צריך לעבור בדרך מהצומת האחת לצומת האחרת.
- צלע יכולה להופיע פעם אחת בלבד במסלול
- אורך המסלול שווה למספר הצלעות בו.
- גובה העץ הוא המסלול הארוך ביותר המחבר את השורש לאחד העלים. גובהו של עץ ריק הוא -1.

לדוגמה בעץ המצויר מימין:

- בין צמתים h ו- e אורך המסלול הוא 3.
- בתת-העץ ששורשו b אורך המסלול הארוך ביותר הוא 4.
- בתת-העץ ששורשו c אורך המסלול הארוך ביותר הוא 4 והוא <u>אינו</u> עובר דרך השורש.
- בעץ כולו, אורך המסלול הארוך ביותר הוא 8.

<u>סעיף א'</u>

המחלקה BTN מייצגת צומת (קודקוד) בעץ בינארי. במחלקה זו:

- משתני עצם `left, right` ו- `data` וכן שיטות המחזירות אותם.
- השיטות `int hight()` ו- `int maxPath()` מחזירות את גובה העץ שצומת (קודקוד) זו היא שורשו ואת אורכו של המסלול הארוך ביותר בתת-העץ. בדוגמה שלמעלה `b.hight()` יחזיר 3 ו- `b.maxPath()` יחזיר 4.

```
class BTN{
    private Object data;
    private BTN left;
    private BTN right;
    public BTN getLeft(){return left;}
    public BTN getRight(){return right;}
    public Object getData(){return data;}
    public int height(){אין צורך להשלים}

    public int maxPath() {
     int maxL = -1;
     int maxR = -1;
     //        א. השלימו בדף התשובות (15 נקודות).
    }
}
```

<u>רמז:</u> המשתנים maxL ו- maxR ייצגו את האורך המסלול הארוך ביותר בתת-העצים השמאלי והימני בהתאמה

<u>הערה:</u> מותר להשתמש בשיטה int Math.max(int, int) המחזירה את הערך הגדול מבין שני הפרמטרים שלה.

<u>סעיף ב':</u>

המחלקה BT מייצגת עץ בינארי. נתייחס לשתי שיטות בלבד:

- `int hight()` - מחזירה את גובה העץ.
- `int maxPath()` - מחזירה את אורכו של המסלול הארוך ביותר בעץ.

```
class BT{
    private BTN root;
    public int height(){ } אין צורך להשלים
    public int maxPath() {
    //          ב. השלימו בדף התשובות (5 נקודות).
    }
}
```

שאלה 2: (30 נקודות)

מונום הוא הביטוי מהצורה $\mathbf{aX^b}$ כאשר a (המקדם) הוא מספר ממשי כלשהו השונה מאפס, ו- b (החזקה) הוא מספר שלם לא שלילי.
פולינום הוא סכום של מונומים כך שאין שני מונומים בעלי אותה חזקה. לדוגמה הביטוי $2X^0+3X^1+4X^2$ הוא פולינום.

בשאלה זו הנכם מתבקשים לממש פולינום כרשימה של מונומים.
נתונה המחלקה List המייצגת רשימה משורשרת של עצמים. במחלקה list קיימות השיטות הבאות:

- `List()` : בונה את הרשימה הריקה.
- `void add(Object x)` : הכנסה של איבר חדש בתחילת רשימה.
- `boolean isEmpty()` : מחזיר אמת אם ורק אם הרשימה ריקה.
- `Iterator iter()` : מחזיר Iterator לרשימה.
- `void filter(Filter f)` :פילטר אשר מוחק מעצם המפתח כל חוליה x המקיימת

f.filter(x.getData()) == false;

להזכירכם:

```
public interface Iterator {
   public boolean hasNext();
   public Object next;
}

public interface Filter {
   public boolean filter(Object x);
}
```

נתונה גם המחלקה Monom .
במחלקה מוגדרים שני משתני עצם:

- coef מייצג את המקדם של המונום (a במונום $\mathbf{aX^b}$).
- pow מייצג את החזקה של המונום. (b במונום $\mathbf{aX^b}$).

במחלקה מוגדרות השיטות הבאות:

- coef() ו- pow() מחזירות את המשתנים המתאימים.
- השיטה add מיישמת חיבור של מונומים בעלי אותה חזקה (הבדיקה שהחזקות שוות באחריות המשתמש)
- השיטה mul מיישמת כפל מונומים.

```
class Monom {
     protected double coef;
     protected int pow;
     public Monom(double t, int p) {coef = t; pow = p;}
     public Monom(Monom m) {this(m.coef, m.pow);}
     public double coef() {return coef;}
     public double pow() {return pow;}
     public void add(Monom m) {coef = coef + m.coef; }
     public void mul(Monom m) {
          coef = coef * m.coef;
          pow = pow + m.pow;
     }
}
```

השתמשו במחלקות List, Monom כדי להשלים את המחלקה Polynom שמייצגת פולינום מעל הממשיים – כלומר אוסף של מונומים בעלי חזקות שונות, שמקדמיהם שונים מ- 0:
במחלקה זו:

- `Polynom()` – בונה היוצר פולינום ריק (ללא מונומים).
- Polynom(Polynom p) בונה המעתיק את הפרמטר העתקה עמוקה (שינוי של הפולינום החדש לא ישנה את p).
- השיטה `del0` void מסירה את כל המונומים בעלי מקדם 0 מהפולינום (עצם המפתח). השיטה משתמשת במחלקה הפנימית Del0.
- השיטה `Object find(Monom m)` מחזירה את ההפניה למונום בעל חזקה כמו `m` אם יש כזה. ניתן להניח ש `m` אינו `null` (אם לא קיים מונום בעל אותה חזקה כמו m השיטה מחזירה `null`).
- השיטה `void add(Monom m)` מיישמת חיבור מונום לפולינום. דוגמה: חיבור של המונום $2X^2$ לפולינום $3X^3+4X^4+2X^2$ נותן את הפולינום $3X^3+4X^4+2X^2$ ו- חיבור של המונום $2X^4$ לפולינום $3X^3+4X^4$ נותן את הפולינום $3X^3+6X^4$. השיטה משנה את עצם המפתח.
- השיטה `void add(Polynom p)` מחברת שני פולינומים משנה את עצם המפתח.
- השיטה `void mul(Monom m)` כופלת את הפולינום במונום m משנה את עצם המפתח.
- השיטה `Polynom mul(Polynom p)` מחזירה פולינום חדש שהוא מכפלת עצם המפתח עם הפרמטר. השיטה אינה משנה את עצם המפתח או את הפרמטר.

שימו לב:

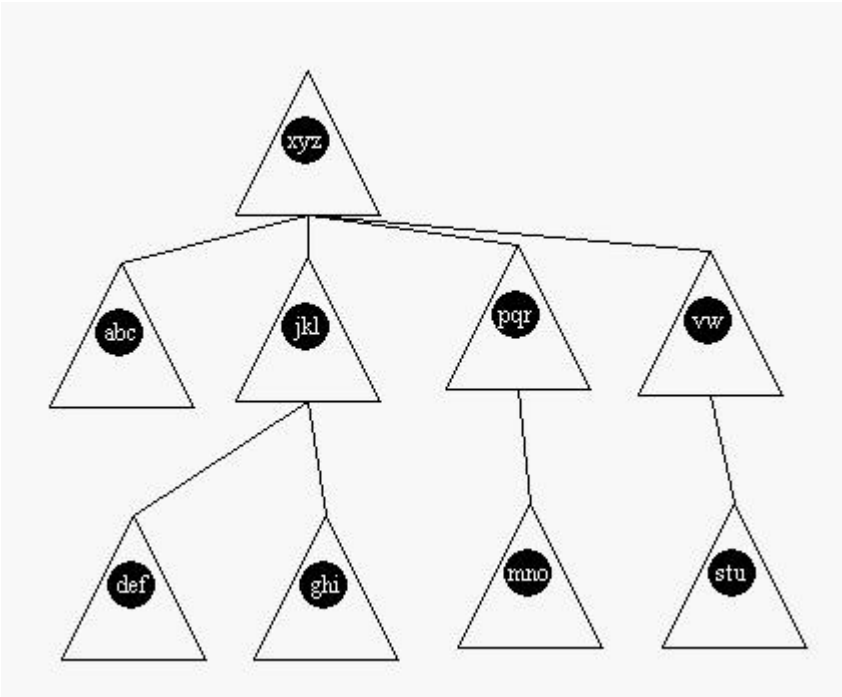
- אסור לשנות את הפרמטרים.
- חשוב לקרוא היטב האם השיטה משנה את עצם המפתח או מחזירה ערך.
- ניתן להניח קלט תקין ללא null, וכן כל מונום כפרמטר הוא תקין כלומר החזקה היא אי שלילית.
- לפולינום אסור שיהיה יותר ממונום אחד מכל חזקה.
- לפולינום אסור שיהיה מונום בעל מקדם 0. באחריותכם לדאוג לכך.

```
public class Polynom extends List {
      public Polynom() {
            super();
      }
      public Polynom(Polynom p) {
            Iterator i = p.iter();
        //      א. השלימו בדף התשובות (4 נקודות).
      }
      public void del0() {
            filter(new Del0());
      }
      private class Del0 implements Filter {
            public boolean filter(Object obj) {
        //      ב. השלימו בדף התשובות (4 נקודות)
            }// filter
       }

      public Object find(Monom m){
            Iterator i = iter();
        //      ג. השלימו בדף התשובות (4 נקודות)
      }

      public void add(Monom m) {
        //      ד. השלימו בדף התשובות (4 נקודות).
      }
      public void add(Polynom p) {
            Iterator i = p.iter();
         //    ה. השלימו בדף התשובות (4 נקודות).
      }
      public void mul(Monom m) {
            Iterator i = iter();
         //    ו. השלימו בדף התשובות (4 נקודות).
      }
      public Polynom mul(Polynom p) {
            Iterator i = p.iter();
        //      ז. השלימו בדף התשובות (6 נקודות)
      }
} // Polynom
```

<u>שאלה 3: 25 נקודות</u>

המחלקה `StringTree` מיצגת עץ (לאו דווקה בינארי) שבצמתיו מאוכסנים עצמים מטפוס String (אף פעם לא `null`). העץ מוגדר על ידי השורש שלו וילדיו (תת-עצים). בשאלה זו ניתן להתיחס לשיטות הבאות של המחלקה:

- השיטה `Object rootString()` מחזירה את המחרוזת המאוכסנת בשורש.
- השיטה `Iterator childrenIterator()` מחזירה איטרטור (ראו הגדרה בסוף השאלה) העובר משמאל לימין על הילדים של העץ (תת-עצים). כלומר בכל קריאה ל- `next` נקבל הפניה לתת-עץ המושרש בבן הבא.
- המחלקה הפנימית `POIterator` היא איטרטור העובר על המחרוזות בעץ `postorder`. כלומר, ראשית נקבל משמאל לימין את המחרוזות המאוכסנות בילדים (תת-העצים) ואחריהם את המחרוזת המאוכסנת בשורש.
- השיטה `postOrderIterator()` במחלקה `StringTree` מחזירה איטרטור מטיפוס `POIterator` העובר על קודקודי עץ `postorder`.



דוגמה: אם המשתנה t מטיפוס StringTree מצביע על העץ שבתמונה, t.rootString() יחזיר את המחרוזת "xyz".
ואם Iterator i = t.childrenIterator()
הרי שניתן לקרוא ל- i.next() ארבע פעמים.
לפני ש- i.hasNext() יתן ערך שלילי.

הפקודות

```
Iterator j = t.childrenIterator();
StringTree child = j.next();
System.out.println(child.rootString());
```

ידפיסו

abc

ו-

```
child.childrenIterator().hasNext();
```

יחזיר false שכן לבן השמאלי אין ילדים.

.

השלימו את המחלקה StringTree. מותר להשתמש לשם כך במחלקה MyStack הממשת את המימשק Stack (הגדרה בסוף השאלה).

מותר להניח שימוש תקין באיטרטור. כלומר באחריות המשתמש שלא לקרוא לשיטה next() מבלי לבדוק קודם לכן אם hasNext() ובאחריותו שלא לקרוא לשיטות של האיטרטור אחרי שיוני העץ. לכן בשאלה זו אין צורך לזרוק exceptions.

```
public class StringTree {
    private Node root;

    public Object rootString() {אין צורך להשלים }
    public Iterator childrenIterator() {אין צורך להשלים }
    private class POIterator implements Iterator {
        private Stack stk;
        public POIterator(StringTree t) {
              stk = new MyStack();
              stk.push(t);
        }
        public boolean hasNext() {
              א. השלימו בדף התשובות (7 נקודות).
                                                          //
        }
        public Object next() {
            Object top = stk.pop();
        //      ב. השלימו בדף התשובות (13 נקודות).
        }
    }
    public Iterator postOrderIterator() {
        //      ג. השלימו בדף התשובות (5 נקודות).
    }
}
```

להזכירכם

```
public interface Stack {
   public boolean isEmpty();
   public boolean push(Object obj);
   public Object pop();
}
```

```
public interface Iterator {
   public boolean hasNext();
   public Object next();
}
```

<u>שאלה 4: (25 נקודות)</u>

תור עדיפויות הוא תור "לא צודק" שבו סדר היציאה של העצמים אינו תלוי בסדר הכניסה אלא בעדיפות שלהם - בעלי העדיפות הגבוהה יצאו ראשונים. בשאלה זו נגדיר שככל שעצם קטן יותר עדיפותו גדולה יותר. לשם הפשטות, נניח שכל העצמים שנכניס לתור הם בני השוואה כלומר מיישמים את הממשק Comparable (מובא בסוף השאלה) ועבור כל שני עצמים A ו- B A.compareTo(B) אינו זורק Exception. יתרה מזו נניח שאין שני עצמים בעלי אותו גודל ולכן גם אין שני עצמים בעלי אותה עדיפות. אין צורך לבדוק הנחות אלו.

בשאלה זו מטרתנו ליישם תור עדיפויות בעזרת עץ חיפוש בינארי.
נתון הממשק Queue שמייצג תור

```
public interface Queue {
        public boolean isEmpty();
        public void enQueue(Object q);
        public Object deQueue();
}
```

נתונה המחלקה BTN מהשאלה הקודמת שמייצגת צומת בעץ בינארי.

<u>סעיף א':</u>

נתונה המחלקה BST המייצגת עץ חיפוש בינארי מאוזן.
במחלקה BST מוגדר בבאי היוצר עץ ריק וכן השיטות הבאות:

- `void add(Comparable data)` - מוסיפה את העצם לעץ החיפוש כך שהוא נישאר עץ חיפוש מאוזן.
- `Comparable find (Comparable key)` - מחפשת עצם השווה ל `key` על העץ ומחזירה הפניה אליו אם הוא נמצא או `null` אם הוא לא נמצא.
- `boolean isEmpty()` - מחזירה אמת אם ורק אם העץ ריק (`root == null`).
- `void remove(Comparable data)` - מוחקת מהעץ עצם השווה (`equals`) ל- `data` (אם יש כזה) תוך שמירה על היותו עץ חיפוש בינארי מאוזן.
- Object min() - מחזירה את הערך הקטן ביותר בעץ החיפוש ולא משנה את העץ.

```
class BST{
  BTN root;
   public BST(){ אין צורך להשלים}
   public void add(Comparable data){ אין צורך להשלים }
   public Comparable find(Comparable key) { אין צורך להשלים }
   public boolean isEmpty() { אין צורך להשלים }
   public void remove(Comparable data) {אין צורך להשלים}
   public Object min() {
            if(root == null) return null;
            BTN answer = root;
            //          א. השלימו בדף התשובות (5 נקודות).
            //  שימו לב שהשיטה היא של העץ ולא של צומת.
          //  רמז: אפשר לפתור בצורה איטרטיבית.
   }
}
```

תזכורת

```
public interface Comparable {
        public int compareTo(Object obj);
}
```

ערך מוחזר חיובי מעיד שעצם המפתח גדול מהפרמטר.

<u>סעיפים ב'-ו':</u>

כעת ברצוננו להשתמש במחלקה BST כדי לכתוב מחלקה (PQ) שמייצגת תור עדיפויות - אוסף של איברים בו האיבר שבראש התור הוא האיבר בעל הערך הקטן ביותר, וממשת את הממשק Queue. במחלקה יהיו הבונים והשיטות הבאים:

- הבונה PQ() יוצר תור ריק.
- הבונה `PQ(PQ q)` מקבל עצם מסוג תור עדיפויות, ובונה תור עדיפויות בעל אותם איברים (הפניות). ניתן להניח ש- `q` אינו `null`. <u>הבונה אינו משנה את הפרמטר</u>.
- השיטה `void enQueue(Object obj)` מכניסה איבר חדש לתור. <u>ניתן להניח קלט תקין ובפרט ש- `obj` אינו `null`.</u>
- השיטה `Object deQueue()` שולפת את האיבר הראשון בתור ומחזירה אותו. אין צורך לטפל (לזרוק `exception` למשל) במקרה שמנסים להפעיל את השיטה על תור ריק.
- השיטה `boolean isEmpty()` מחזירה אמת אם ורק אם התור ריק. <u>עצם המפתח אינו משתנה בעקבות הפעלת השיטה.</u>
- השיטה `boolean equals(Object obj)` מחזירה אמת אם הפרמטר הוא תור עדיפויות בעל אותם איברים (האיברים המתאימים עומדים ביחס `equals` באותו סדר כמו עצם המפתח). <u>עצם המפתח והפרמטר אינם משתנים בעקבות הפעלת השיטה.</u>

השלימו את הגדרת המחלקה.

```
public class PQ extends BST implements Queue {
    public PQ() {
      super();
    }
    public PQ(PQ q) {
          //      ב. השלימו בדף התשובות (7 נקודות).
    }
    public void enQueue(Object obj) {
         //      ג. השלימו בדף התשובות (נקודה אחת).
    }
    public Object deQueue() {
          //      ד. השלימו בדף התשובות (2 נקודות).
    }
    public boolean equals(Object obj) {
        //      ה. השלימו בדף התשובות (10 נקודות).
    }
}
```

<u>שימו לב:</u>

- ניתן להניח שהפרמטרים של השיטות אינם null.
- אסור לשנות את הפרמטרים שמועברים לשיטות
- אסור שהשיטה equals תשנה את עצם המפתח.