תכנות מונחה עצמים - OOP
פתרון מועד ב

1. (4%)
מה יהיה הפלט של הקוד הבא:

```
interface hasIterator {
   public Iterator iterator();
}

class MyList extends LinkedList implements hasIterator {}

class Check {
   static boolean hasDup(hasIterator dataStructue) {
     for(Iterator it1 = dataStructue.iterator(); it1.hasNext(); ) {
        Object v = it1.next();
        int count = 0;
        for(Iterator it2 = dataStructue.iterator(); it2.hasNext(); ) {
              if(it2.next().equals(v))
                 ++count;
        }
        if (count>1)
              return true;
     }
     return false;
   }

   static public void main(String[] args) {
     int[] array = {3,4,2,4,5};
     MyList l = new MyList();
     for(int i=0; i<array.length; ++i)
        l.add(new Integer(array[i]));
     if(hasDup(l))
        System.out.println("has duplication");
     else
        System.out.println("no duplication");
   }
}
```

פלט: ___________________________

תשובה: has duplication

2. (5%)
הפונקציה hasDup היא כללית מאוד, היא עובדת על כל מבנה נתונים המספק איטרטור.
האם לכלליות הזאת יכל להיות מחיר בזמן ריצה? הסבירו.

____

____

תשובה: כן. ישנם מבני נתונים מסוימים עליהם ניתן לבצע בדיקת כפילויות בצורה יעלה יותר. רשימה של מספרים לדוגמה, יעיל יותר למיין ואז לבדוק כפילויות.

3. (5%)

הקוד הבא הוא ניסיון שיפור כושל של הפונקציה hasDup:

```
static boolean hasDup(hasIterator dataStructue) {
    for(Iterator it1 = dataStructue.iterator(); it1.hasNext(); ) {
        Object v = it1.next();
        int count = 0;
        for(Iterator it2 = it1; it2.hasNext(); ) {
            if(it2.next().equals(v))
                return true;
        }
    }
    return false;
}
```

הסבירו איזו בדיקה מבצע קוד זה ומדוע אין זו הבדיקה המבוקשת.

____

____

תשובה:

מכיוון ש it2 איננו העתק של it1 אלא שם נירדף לו, הלולאה הפנימית גורמת להתקדמות הלולאה החיצונית. בסופו של דבר, הפונקציה תבדוק כפילויות של האבר הראשון בלבד.

4. (5%)

נתון הקוד הבא המתאר עץ בינארי ממוין (רלוונטי לשאלות 4-9):

```
class Tree {
    private static class Node {
        int _data;
        Node _ls=null,_rs=null;
        Node(int data) { _data = data; }
    }
    private Node _root=null;
    public void insert(int data) {
        if(_root == null) {
            _root = new Node(data);
            return;
        }
        Node p = _root;
        while(true) {
            if(data < p._data)
                if(p._ls == null) {
                    p._ls = new Node(data);
                    break;
                } else
                    p = p._ls;
            else
                if(p._rs==null) {
                    p._rs = new Node(data);
                    break;
                } else
                    p = p._rs;
        }
    }
    private String toString(Node p) {
        if (p==null)
            return "n";
        return "("+toString(p._ls)+","+p._data+","+toString(p._rs)+")";
```

```
    }
    public String toString() {
      return toString(_root);
    }
    static public void main(String[] args) {
      int[] array = {4,1,3,2};
      Tree t = new Tree();
      for(int i=0; i<array.length; ++i)
          t.insert(array[i]);
      System.out.println(t);
    }
}
```

מה יהיה פלט ה - main הנתון?

____

תשובה:

((n,1,((n,2,n),3,n)),4,n)

5. (5%)

נתון מימוש אלטרנטיבי לפונקצית ה - insert:

```
    private void insert(int data, Node p) {
      if (data < p._data)
          if(p._ls == null)
                  p._ls = new Node(data);
          else
                  insert(data,p._ls);
      else
          if(p._rs == null)
                  p._rs = new Node(data);
          else
                  insert(data,p._rs);
    }

   public void insert(int data) {
      if(_root == null)
          _root = new Node(data);
      else
          insert(data,_root);
    }
```

האם מימוש זה עדיף על המימוש המקורי? האם המימוש המקורי עדיף? אולי אין העדפה? הסבירו את תשובתכם.

____

____

____

תשובה:

המימוש המקורי עדיף. המימוש האלטרנטיבי משתמש ברקורסיית זנב בשביל ליישם אלגוריתם שהגרסה המקורית ממשה בצורה איטרטיבית. רקורסיית זנב בזבזנית בזיכרון ובזמן הריצה לעומת קוד איטרטיבי שקול.

6. (5%)

הקוד הבא אמור להוות אלטרנטיבה נוספת למימוש - insert:

```
private void insert(int data, Node p) {
     if(p == null) {
        p = new Node(data);
        return;
     }
     if (data < p._data)
        insert(data,p._ls);
     else
        insert(data,p._rs);
   }

   public void insert(int data) {
     insert(data,_root);
   }
```

האם הוא באמת מהווה אלטרנטיבה תקינה? האם החלפת המימוש המקורי של insert במימוש זה תגרור שינוי בפלט התוכנית? אם כן, מה יהיה הפלט החדש?

תשובה:

המימוש המוצע איננו מהווה אלטרנטיבה תקינה. מכיוון של - insert הפרטית מועבר רק <u>העתק</u> של מצביע לקודקוד, כל שינוי בהצבעה שלו, לא ישפיע על הפונקציה הקוראת. בגלל סיבה זו, העץ יישאר ריק לחלוטין והפלט החדש יהיה: n.

7. (13%)

הוסיפו לקוד המקורי פונקציה בשם getAllSubtrees, המאפשרת קבלת כל תתי העצים של העץ. הפונקציה תחזיר ArrayList של עצים המהווים <u>העתק</u> של תתי העצים של העץ. מכיוון שמדובר בהעתקים, לאחר קבלת אוסף תתי העצים, כל שינוי של העץ המקורי לא ישפיע על האוסף ולהפך.

הקוד הבא מדגים שימוש בפונקציה שעליכם לכתוב:

```
static public void main(String[] args) {
     int[] array = {30,10,40,20};
     Tree t = new Tree();
     for(int i=0; i<array.length; ++i)
        t.insert(array[i]);
     ArrayList allSubs = t.getAllSubtrees();
     System.out.print("Original:\n"+t+"\nSubtrees:\n");
     t.insert(17777); // shouldn't effect allSubs
     for(int i=0; i<allSubs.size(); ++i)
        System.out.print(allSubs.get(i)+" | ");
   }
```

פלט:

```
Original:
((n,10,(n,20,n)),30,(n,40,n))
Subtrees:
(n,10,(n,20,n)) | (n,20,n) | ((n,10,(n,20,n)),30,(n,40,n)) | (n,40,n) |
```

סדר תתי העצים איננו מחייב.

<u>הערה</u>: תת עץ הוא עץ המכיל קודקוד מהעץ המקורי ואת כל צאצאיו. לפי הגדרה זו, עץ ריק איננו מהווה תת עץ.

:תשובה

```
private Node getSubCopy(Node p) {
      if (p==null)
          return null;
      Node ret = new Node(p._data);
      ret._ls = getSubCopy(p._ls);
      ret._rs = getSubCopy(p._rs);
      return ret;
   }

   private Tree(Node root) {_root = root;} // requires adding: Tree() {}

   private Tree getSubtreeCopy(Node p) {
      return new Tree(getSubCopy(p));
   }

   private void getAllSubtrees(ArrayList acum, Node p) {
      if(p==null)
          return;
      getAllSubtrees(acum,p._ls);
      acum.add(getSubtreeCopy(p));
      getAllSubtrees(acum,p._rs);
```

```
    }

    public ArrayList getAllSubtrees() {
      ArrayList ret = new ArrayList();
      getAllSubtrees(ret,_root);
      return ret;
    }
```

8. (7%)

כתבו פונקציה המאפשרת לשמור את מבנה הנתונים של תתי העצים מהשאלה הקודמת, בקובץ. אתם יכולים להניח את קיום הפונקציה:

```
class Tree {
…
  public void save(FileWriter fw) throws Exception {…}
}
```

המקבלת אובייקט כתיבה לקובץ ומקודדת לתוכו את תוכן העץ.

על הפונקציה שלכם להקרא saveSubs והיא תופעל באופן הבא:

```
    Tree t = new Tree();
    …
    ArrayList allSubs = t.getAllSubtrees();
    saveSubs(allSubs,"subs.txt");
```

כאשר sub.txt הוא שם הקובץ בו ישמר מבנה הנתונים.

____

____

____

____

____

____

____

תשובה:

```
public static void saveSubs(ArrayList subs,String name) throws Exception {
    FileWriter fw = new FileWriter(name);
    for(int i=0; i<subs.size(); ++i)
       ((Tree)subs.get(i)).save(fw);
    fw.close();
  }
```

9. (7%)

כתבו פונקציה המסוגלת לקרוא את מבנה הנתונים שנישמר ע"י הפונקציה מהשאלה הקודמת. ניתן להניח את קיום הפונקציה:

```
class Tree {
…
 public void load(FileReader fr) throws Exception {…}
}
```

הקוראת קובץ שנכתב ע"י פונקצית ה - save שאת קיומה הנחתם בשאלה הקודמת.

לפונקציה יש לקרוא loadSubs והיא תופעל באופן הבא:

```
    Tree t = new Tree();
    …
    ArrayList allSubs = t.getAllSubtrees();
```

```
saveSubs(allSubs,"subs.txt");
ArrayList sub2 = loadSubs("subs.txt");
```

תשובה:

____

____

____

____

____

____

____

תשובה:

```
public static ArrayList loadSubs(String name) throws Exception {
    FileReader fr = new FileReader(name);
    ArrayList ret = new ArrayList();
    while(fr.ready()) {
       Tree tmp = new Tree();
       tmp.load(fr);
       ret.add(tmp);
    }
    return ret;
  }
```

10. (5%)

הסבירו מדוע הקוד הבא לא יעבור קומפילציה:

```
class A {
   static B foo() { return new B(); }
   class B{}
}
```

הסבר:

____

____

____

תשובה:

מכיוון ש B היא מחלקה פנימית לא סטטית, אובייקט מסוגה אמור להחזיק מצביע לאובייקט שיצר אותו. הפונקציה foo היא סטטית וככזאת לא מכירה אף אובייקט ספציפי ואין באפשרותה לתת לאובייקט B שייצרה את המצביע הדרוש לו.

11. (5%)

מדוע Java לא מוכנה לקמפל את הקוד הבא ואיזה תועלת מפיק המתכנת מתכונה זו של Java?

```
class A {
   private class B{
   }
}
class Check {
   static public void main(String[] args) {
     A.B b;
   }
```

```
}
```

הסבר:

תשובה:

מכיוון ש B הוגדרה מחלקה פנימית פרטית, הטיפוס B אינו מוכר מחוץ ל - A. הדבר מאפשר אנקפסוציה ברמת טיפוס. ניתן להחליף את B בטיפוס אחר ולדעת ששום קוד חיצוני לא יושפע מהשינוי.

12. (5%)

הקוד הבא מתאר טיפוס של טבלת גיבוב בגודל קבוע (רלוונטי לשאלות 12-16):

```
class MyHashTable {
   public final int N = 10;
   private ArrayList _array=new ArrayList();

   static private class Pair {
     Object _key, _value;
     Pair (Object key, Object value) { _key = key; _value = value; }
   }

   public MyHashTable() {
     for(int i=0; i<N; ++i)
        _array.add(new LinkedList());
   }

   private int hash(int n) { return n%N; }

   public void put(Object key, Object value) {
     LinkedList list = (LinkedList)_array.get(hash(key.hashCode()));
     list.add(new Pair(key,value)); //   מוסיף לסוף הרשימה
   }

   public Object get(Object key) {
     LinkedList list = (LinkedList)_array.get(hash(key.hashCode()));
     for(Iterator it= list.iterator(); it.hasNext();) {
        Pair p = (Pair)it.next();
        if(p._key.equals(key))
          return p._value;
     }
     return null;
   }
public String toString() {
     String str = "";
     for(int i=0; i<N; ++i) {
        Iterator it=((LinkedList)_array.get(i)).iterator();
        if(!it.hasNext())
              continue;
        while(it.hasNext())
              str+=((Pair)it.next())._value+"->";
        str+="\n";
     }
```

```
        return str+"======\n";
    }

    static public void main(String[] args) {
        MyHashTable table = new MyHashTable();
        int[] ids = {13124854,2003433,33602314};
        String[] names = {"Yosi","Pesi","Moshe"};
        for(int i=0; i<3; ++i)
            table.put(new Integer(ids[i]),names[i]);
        System.out.print(table);
    }
}
```

הערה: ה - hashCode של אובייקט מסוג Integer הוא ערכו.

מה יהיה הפלט של ה main הנתון?

____

____

____

____

תשובה:

Pesi->
Yosi->Moshe->
======

13. (4%)
האם אכסון כאלף שמות של אנשים לפי מפתח תעודת הזהות שלהם, דורש שימוש בטבלת גיבוב או שאולי היה מוטב להשתמש במערך רגיל. הסבירו תשובתכם.

____

____

תשובה:

המקרה מתאים לשימוש בטבלת גיבוב. אוסף המפתחות האפשריים גדול בהרבה ממספר האנשים שישמרו במבנה הנתונים. שימוש במערך רגיל היה גורם לבזבוז מקום רב.

14. (4%)
נתונה המחלקה MyInteger:

```
class MyInteger {
    public MyInteger(int n) {}
    public int hashCode() { return 1;}
    public boolean equals(Object o) { return true; }
}
```

מה היה פלט התוכנית אילו היינו משנים את main להשתמש במחלקה MyInteger במקום במחלקה - Integer ?

____

____

____

תשובה:

Yosi->Pesi->Moshe->
======

15. (13%)
הוסיפו את הקוד הדרוש על מנת שטבלת הגיבוב תספק איטרטור. אתם יכולים לבחור את סדר הסריקה כרצונכם, אך צריך לעבור על כל אבר בדיוק פעם אחת. פונקצית next של האיטרטור תחזיר את הערך הנוכחי ולא את המפתח.
לאחר הוספת הקוד שלכם, ניתן יהיה להוסיף את הקוד הבא לסוף ה - main ולקבל את שלושת השמות בסדר כלשהו:

```
for(Iterator it = table.iterator(); it.hasNext(); )
    System.out.print(it.next()+" ");
```

תשובה:
להוסיף בתוך class MyHashTable:

```
private class HashIterator implements Iterator {
    int _i=0;
    Iterator _it = ((LinkedList)(_array.get(0))).iterator();

    private void nextLine() {
        while(!_it.hasNext()) {
                ++_i;
                if(_i==N)
```

```
                return;
            _it = ((LinkedList)(_array.get(_i))).iterator();
        }
    }
    public HashIterator() {
        nextLine();
    }
    public Object next() {
        Object ret =  ((Pair)_it.next())._value;
        if(!_it.hasNext())
            nextLine();
        return ret;
    }
    public boolean hasNext() { return _i < N; }
    public void remove() {} // just because of the Iterator interface
  }

public Iterator iterator() {return new HashIterator(); }
```

16. (4%)

נניח שהיינו מוסיפים לאיטרטור פונקציה המאפשרת לשנות את המפתח הנוכחי מבלי לשנות שום נתון דבר אחר במבנה הנתונים.

א. זו הייתה יכולה להיות תוספת שימושית מאד למקרה של שינוי תעודת זהות למשל.
ב. זה היה פוגם בעקביות הפנימית של מבנה הנתונים ולא מאפשר לו לפעול.
ג. קל מאוד להוסיף פונקציה כזאת. היא לא תזיק אך ספק אם תועיל.
ד. הוספת פונקציה כזו מצביעה על תכנות לקויי מבחינת אנקפסולציה: אין לחשוף את המפתח בפני המשתמש בטבלה.

תשובה:
ב. הדבר עלול לגרום למפתח להיות במקום שאינו מתאים לו מבחינת הגיבוב.

17. (4%)

האם מבנה נתונים צריך להחזיק הצבעות לאובייקטים המקוריים שניתנו לו או להחזיק העתקים שלהם?

א. תלוי בצרכים, לכל אופציה יש יתרון בהקשר מסוים.
ב. תמיד כדאי להחזיק העתק כי אז אין תלות בשינויים שנעשים באובייקטים מאוחר יותר.
ג. תמיד כדאי להחזיק הצבעות מכיוון שהדבר חוסך זיכרון וזמן ריצה.
ד. תמיד כדאי להחזיק הצבעות מכיוון שאז אפשר לגרום לשינוי באובייקט מסוים להשתקף בכל מבני הנתונים שאליהם הוא שייך.

תשובה: א. למרות שכל הטענות המופיעות בסעיפים האחרים הם נכונות. איחוד של כולן מוביל ל - א.