מבחן מסכם מועד-א' ב"מבוא למדעי המחשב" 201-1-101-1

סמסטר א' תשס"א
1.2.2001

פרופ' אורי אברהם
פרופ' דניאל ברנד
ד"ר שמואל ספרוני
ד"ר מיכאל קודיש

משך הבחינה שעתיים וחצי.
חומר עזר אסור.
אין להשתמש במחשבון.

במבחן זה 8 שאלות המאפשרות לצבור עד 105 נקודות. מותר לפתור את כל השאלות או את חלקן.
הציון המרבי במבחן הוא 100.

אנא רשמו את תשובותיכם בדף התשובות בלבד. הקפידו לרשום בדף התשובות גם את מספר הנבחן ומספר החדר שבו אתם נבחנים. המחברת שקיבלתם היא **מחברת הטיוטה והיא לא תימסר כלל לבדיקה**.
בסיום הבחינה נאסוף אך ורק את דף התשובות.

בהצלחה

שאלה 1 (30 נקודות)

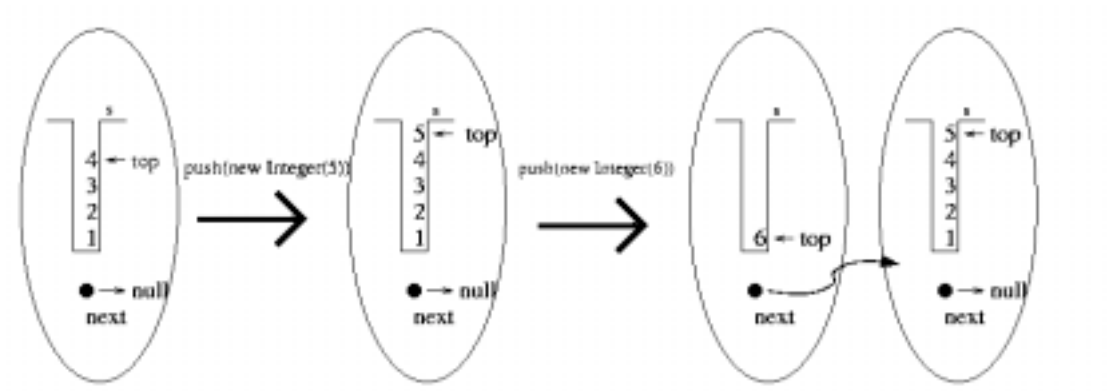
במהלך הקורס תיארנו שני מימושים לממשק (interface) המחסנית: הראשון -- *מחסנית-מערך* (StackAsArray) -- בעזרת מערך של עצמים, והשני -- *מחסנית-רשימה* (StackAsList) -- בעזרת רשימה משורשרת של עצמים.

בשאלה זו נבנה מימוש נוסף עבור ממשק המחסנית שיקרא *רב-מחסנית* (MultiStack). המימוש דומה לזה של מחסנית-מערך מבחינה זו שהאיברים מוחזקים במערך של עצמים. ההבדל הוא שכאשר מתמלא המערך ניתן להוסיף איברים נוספים על-ידי הגשמת מחסנית-מערך נוספת שאותה משרשרים לקודמת במבנה של רשימה משורשרת. בשאלה זו נניח שהמערכים במבנה מכילים חמישה איברים כל אחד.

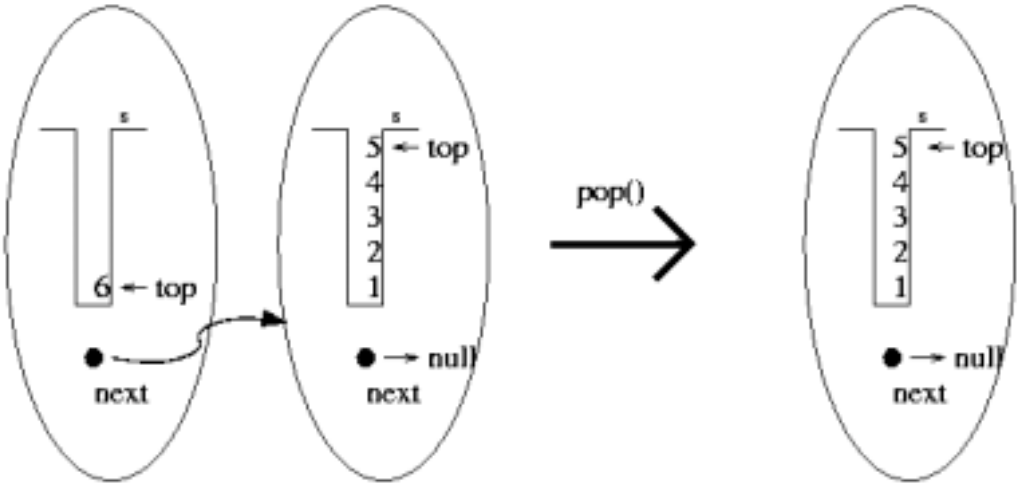
המימוש של רב-מחסנית למעשה אינו כה מורכב, מכיוון שניתן לתאר אותה כעצם מסוג מחסנית-רשימה המאגד בתוכו אוסף של מחסניות-מערך, שבתוכם נמצאים איברי הרב-מחסנית. כותרת המחלקה תיראה כדלקמן:

```
public class MultiStack extends StackAsList
                        implements StackInterface
```

הוספת איבר x לרב-מחסנית m מתבצעת במידת האפשר על מחסנית-המערך העליונה ב-m. במידה ש-m ריקה ממחסניות, או שמחסנית-המערך העליונה ב-m מלאה, מוסיפים מחסנית-מערך חדשה ל-m ולתוכה מוסיפים את x. הציור הבא מדגים שתי פעולות הוספה (של האיבר 5 והאיבר 6) על רב-המחסנית הנתונה.



הוצאת איבר מרב-מחסנית m (לא ריקה) מתבצעת ממחסנית-המערך העליונה ב-m, אך מקפידים לא להשאיר מחסניות-מערך ריקות על m. הוצאת איבר מרב-מחסנית ריקה ייתן ערך null. הציור הבא מדגים פעולת הוצאה של האיבר 6 מרב-המחסנית הנתונה.



להלן תמצאו הגדרה חלקית של המחלקה MultiStack ואת הגדרת ממשק המחסנית ומימושיו בעזרת מערך ורשימות משורשרות. בסעיפים א' ו-ב' של שאלה זו אתם מתבקשים להשלים את השיטות עבור הוספה (push) והוצאה (pop) במחלקה MultiStack.

זה ממשק המחסנית שאותו אתם צריכים לממש:

```
public interface StackInterface{
        boolean push(Object a);
        Object pop();
        Object peek();
        boolean isEmpty();
        boolean isFull();
} // interface StackInterface
```

זה מימוש חלקי שאותו תשלימו:

```
public class MultiStack extends StackAsList
                            implements StackInterface{
    public MultiStack(){
        super();
    }
    public boolean push(Object a){
      /***
                                        סעיף א': השלם (20 נקודות)
      ***/
    }
    public Object pop(){
      /***
                                        סעיף ב': השלם (10 נקודות)
      ***/
    }
    public Object peek(){
      /**
                                    הניחו שאת שיטה זו מימשנו עבורכם
      **/
    }

    //    public boolean isEmpty(){ ** is inherited **}
    //    public boolean isFull(){ ** is inherited **}
}
```

זה המימוש של מחסנית-מערך (להזכירכם)

```
    public class StackAsArray implements StackInterface{

            Object[] s;
            int top;

            public StackAsArray( ){
                    s = new Object[5];
                    top = 0;
            }
            public boolean push(Object a){
                    if (isFull()) {
                            System.out.print("Stack is full");
                            return false;
                    }
                    else{   s[top] = a;
                            top = top +1;
                            return true;
                    }
            }
```

```
        public Object pop(){
                Object r = null;
                if (!isEmpty()){
                        r = s[top - 1];
                        top = top - 1;
                }
                return r;
        }

        public Object peek(){
                Object r = null;
                if (!isEmpty()) r = s[top - 1];
                return r;
        }

        public boolean isEmpty(){
                return top == 0;
        }

        public boolean isFull(){
                return top == s.length;
        }
 } // class StackAsArray
```

זה המימוש של מחסנית-רשימה (להזכירכם):

```
public class StackAsList extends List
                 implements StackInterface{
   public StackAsList(){
       super();
   }
    public boolean push(Object a){
        addAtHead(a);
        return true;
   }
    public Object pop(){
        Object Value = null;
        if (!isEmpty()){
            Value = first.get_data();
            deleteHead();
        }
        return Value;
   }
    public Object peek(){
        Object Value = null;
        if (!isEmpty())
            Value = first.get_data();
        return Value;
   }

    public boolean isEmpty(){
        return (first == null);
    }

    public boolean isFull(){
        return false;
    }
}  // class StackAsList
```

זה המימוש של רשימה משורשרת (להזכירכם):

```
public class List{
        public Node first;

        List(){
                first = null;
        }

        public void addAtHead(Object data){
                first = new Node(data, first);
        }

        public void deleteHead(){
        // we assume that this method is called only
        // when first is not equal to null
                first = first.get_next();
        }

} // class List
```

זה המימוש של חוליה (להזכירכם):
שימו לב שבחלק מקבוצות ההרצאה מחלקה זו הופיעה בשם Link.

```
public class Node{
  public Object data;
  public Node next;

   Node(Object data, Node next){
     this.data = data;
     this.next = next;
   }
    public Object get_data(){return data;}
    public Node get_next(){return next;}
    public void set_next(Node a){next = a;}
}// Node
```

שאלה 2: (15 נקודות)

נתונה המחלקה MyStack המממשת את הממשק StackInterface אשר מוגדר בעמוד 3. אופן המימוש של המחלקה MyStack אינו ידוע.

ב- java ניתן להרחיב מחלקה רגילה על-ידי מחלקה מופשטת. מטרת המחלקה המופשטת FilterStack המובאת כאן היא לאפשר "סינון" של איברי מחסנית כדי לסלק ממנה את כל אותם האיברים שאינם עומדים בתנאי סינון. תנאי הסינון מוגדר בעזרת שיטה מופשטת filter שתפקידה לבחון איבר b ולהחזיר ערך false אם (ורק אם) יש לסלק את b מהמחסנית. שאר האיברים ישארו לפי סדר הופעתם המקורי. לדוגמא: נוכל להשתמש בשיטה זו כדי להשאיר במחסנית של מספרים רק את הערכים החיוביים.

סעיף א: השלימו את גוף השיטה filtering( ) כך שתעבור על איברי המחסנית ותסלק את אלו שלא עברו את תנאי הסינון.

```
public abstract class FilterStack extends MyStack {
   public FilterStack {super( );}
   public abstract boolean filter(Object b);

   public void filtering() {
   /***
                              סעיף א': השלם (10 נקודות)
   ***/
   }
}
```

סעיף ב: כיתבו את המחלקה StringStack, המרחיבה את FilterStack, כך שהפעלת השיטה filtering( ) שבה תשאיר במחסנית רק עצמים מטיפוס String.

```
public class StringStack extends FilterStack {
  public StringStack() {super();}
   /***
                              סעיף ב': השלם (5 נקודות)
    ***/
}
```

שאלה 3: (10 נקודות)
מה תדפיס התכנית הבאה?

```
public class What{
  public static void process(int[] a){
    boolean is = true;
    int i = a.length - 1;
    while( is && i >= 1){
      is = false;
      for( int j = 0; j < i; j = j + 1)
        if( a[j] > a[j+1]){
          int temp = a[j];
          a[j] = a[j + 1];
          a[j + 1] = temp;
          is = true;
        }
        i = i - 1;
    }
  }

  public static void main(String[] args){
    int[] arr = {3,9,2,6,4,7,1};
    process(arr);
    for( int i = 0; i < arr.length; i = i + 1)
      System.out.print(arr[i] + " " );
  }
}// What
```

בשאלות 4-5 נתייחס לקוד הבא:

```
class A {
  int _x;
```

```
  public A(int i) {_x = i;}
  public String toString() {return "1";}
}

class B extends A {
  public B(int i) {super(i);}
  public String toString() {return "2";}
  public A convertToA() {return new A(_x);}
}

class C{}
```

<u>שאלה 4:</u> (5 נקודות)

רשום בדף התשובות מה תדפיס התכנית הבאה:

```
public class Exam5 {
  public static void main(String[] a) {
    A s1 = new A(10);
    A s2 = new B(20);
    B s3 = new B(30);
    System.out.print(s1);
    System.out.print(s2);
    System.out.print(s3);
    System.out.print((A)s3);
    System.out.print(s3.convertToA());
  }
}
```

<u>שאלה 5:</u> (5 נקודות)

התייחס לתכנית הבאה:

```
public class Exam {
  public static void main(String[] a) {
    Object s5 = new A(5);
    C s4 = (C)s5;
  }
}
```

א. התכנית תעבור קומפילציה, אך בזמן ריצה תיגרם שגיאת המרה (casting).

ב. ההמרה תגרום לשגיאה בזמן קומפילציה, שכן לא ניתן להמיר מטיפוס A ל-C.

ג. אילו s5 היה משתנה מטיפוס A (במקום Object) התכנית לא היתה עוברת קומפילציה, שכן אז היה ברור שלא ניתן לבצע את ההמרה כבר בזמן בדיקת הטיפוסים של הקומפיילר.

ד. התכנית תעבור קומפילציה ותרוץ ללא שגיאות, שכן המחלקה C היא מקרה פרטי של המחלקה A.

ה. התכנית לא עוברת קומפילציה כי אסור להגדיר משתנה (פונה) מטיפוס של מחלקה מופשטת (אבסטרקטית) ו-Object היא מחלקה מופשטת (אבסטרקטית).

ו. אף תשובה לא נכונה.

<u>שאלה 6</u> (15 נקודות)

נתונים שני מערכים של מספרים שלמים (integer) arr1 ו- arr2. שני המערכים ארוכים אך arr2 ארוך בהרבה מ- arr1. למשל, ניתן להניח שהאורך של arr2 הוא כמו הריבוע של האורך של arr1. פעולה

שכיחה, שאותה נתבקש לבצע מספר רב של פעמים (להלן "*הפעולה*"), היא הפעולה הבאה: בהינתן מספר שלם d, מצא איבר x ב- arr1 ואיבר y ב- arr2 כך ש- $x-y=d$.

אזי:

א. אם שני המערכים ממוינים אזי ניתן לממש את הפעולה על-ידי חיפוש בינארי פשוט על ההפרש.

ב. ניתן לממש את הפעולה בעזרת לולאה כפולה שבוחנת את כל זוגות האיברים (אחד מכל מערך). בשיטת מימוש זו, אין זה משנה אם המערכים יהיו ממוינים לפני ביצוע החיפוש.

ג. אם נמיין מראש את שני המערכים, אזי נוכל לממש את הפעולה על-ידי לולאה שבוחנת את איברי אחד המערכים ולכל אחד מהם מבצעת חיפוש בינארי עבור ערך מתאים (בהתאם ל-d) במערך השני.

ד. ניתן לשפר את המימוש המתואר באפשרות ג' ע"י כך שנמיין רק אחד משני המערכים, ואין זה משנה אם יהיה זה המערך שבוחנים את כל איבריו או המערך השני. (התייחס לשיפור מבחינת הזמן שיידרש לכל הרצה עתידית של שיטת החיפוש. התעלם מהעלות החד-פעמית של המיון.)

ה. ניתן לשפר את המימוש המתואר באפשרות ג' ע"י כך שנמיין את arr1 לפי סדר עולה ואת arr2 לפי סדר יורד. (התייחס לשיפור מבחינת הזמן שיידרש לכל הרצה עתידית של שיטת החיפוש. התעלם מהעלות החד-פעמית של המיון.)

ו. ניתן לשפר את המימוש המתואר באפשרות ג' ע"י כך שנמיין רק אחד משני המערכים. עדיף שנמיין רק את arr2 על אף שהוא יותר ארוך ולכן דורש הרבה יותר פעולות מאשר למיין את arr1. (התייחס לשיפור מבחינת הזמן שיידרש לכל הרצה עתידית של שיטת החיפוש. התעלם מהעלות החד-פעמית של המיון.)

<u>שאלה 7</u> (15 נקודות)

האלגוריתם הבא מוצע לצורך מיון: בהינתן מערך arr של שלמים באורך n בצע את הפעולות הבאות:

1. החלף את האיברים arr[0] ו-arr[k], כאשר k הנו מספר האיברים מבין arr[1], arr[2],…, arr[n-1] הקטנים מ- arr[0].

2. החלף את האיברים arr[1] ו-arr[k+1], כאשר k הנו מספר האיברים מבין arr[2], arr[3],…, arr[n-1] הקטנים מ- arr[1].

3. המשך עבור `i= 2,3, ,n-2` באותו אופן. כלומר, החלף את האיברים arr[i] ו-arr[k+i], כאשר k הנו מספר האיברים מבין arr[i+1],…, arr[n-1] הקטנים מ- arr[i].

אזי:

א. האלגוריתם המוצע לא תמיד פועל כשורה.

ב. האלגוריתם המוצע פועל נכון כאשר כל איברי המערך arr שונים זה מזה.

ג. האלגוריתם המוצע עובד כשורה ומבצע בפועל אותן החלפות איברים כמו אלגוריתם המיון על-ידי הכנסה (insertion sort).

ד. האלגוריתם המוצע עובד כשורה ומבצע בפועל אותן החלפות איברים כמו אלגוריתם המיון על-ידי בחירה (selection sort).

ה. האלגוריתם המוצע עובד כשורה, אך הוא שונה מאלגוריתמי המיון שסקרנו בכיתה.

ו. אף לא אחת מהתשובות לעיל.

<u>שאלה 8</u> (10 נקודות)

נתונות שלוש מחלקות: A, B ו- UseAB. המחלקה B מרחיבה את המחלקה A.
הבונים (היחידים) של A ו- B נראים כך:

```
A(int n){
   for (int i=0; i<n; i=i+1)
      new A(i);
   System.out.println("mavo");
}
```

```
B(int n){
   super(n);
   for (int i=0; i<n; i=i+1)
      new B(i);
}
```

בשיטה main המופיעה במחלקה UseAB מופיעה השורה:

```
B b = new B(4);
```

מספר השורות המודפסות בעקבות משפט זה הנו:

א. 16 ב. 31 ג. 32
ד. 48 ה. 64 ו. אף לא אחת מהתשובות לעיל.