תאריך הבחינה: 5.3.2013

שמות המרצים: פרופ' משה זיפר
מר אילן כדר
ד"ר פז כרמי
מר עדי סוויסה
פרופ' מייק קודיש
ד"ר צחי רוזן

שם הקורס: מבוא למדעי המחשב

מספר הקורס: 202-1-1011

שנה: 2013 סמסטר: א' מועד: **ב**

משך הבחינה: **2.5 שעות**

חומר עזר: **אסור**

מבוא למדעי המחשב מועד ב

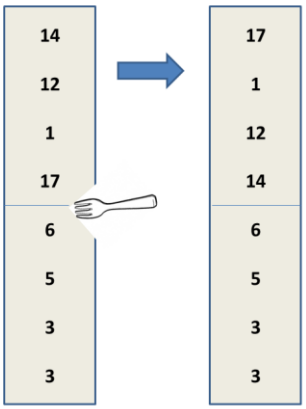
אנא קיראו היטב את ההוראות שלהלן:

- במבחן זה 4 שאלות. ענו על כל השאלות. בשאלות בהן לא מצוין אחרת, ניתן לבחור בפתרון רקורסיבי או פתרון שאינו רקורסיבי, לבחירתכם.
- רשמו את תשובותיכם **בדפי 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.
- **שימו לב:** החשיבות העליונה היא **לנכונות** הקוד. מאידך, **יעילות, סגנון וכתיבה ברורה** חשובים גם הם, ולכן תשובה יעילה ומסוגננת תזכה בציון גבוה יותר.
- בשאלות התכנות, מספר השורות העומדות לרשותכם בדף התשובות רומז על אורך הקוד המקסימלי הנדרש. הקפידו על כתב יד ברור. **תשובות מסורבלות או ארוכות מדי לא יזכו בניקוד מלא**. כתבו פתרון לשאלה קודם במחברת הטיוטה ורק לאחר מכן העתיקו פתרון נקי לדף התשובות.
- בכל השאלות, אין להוסיף פונקציות עזר אלא אם נאמר במפורש שמותר.
- אם יש סעיף בשאלה המסתמך על סעיף אחר, מותר להשתמש בו גם אם לא פתרתם את הסעיף האחר.
- **הקפידו לרשום בכל דפי התשובות גם את מספר הנבחן ומספר החדר שבו אתם נבחנים.**
- **במידה ואינכם יודעים את התשובה לסעיף כלשהו, רשמו "לא יודע/ת" (במקום תשובה) ותזכו ב- 20% מניקוד הסעיף (מעוגל מטה). אם רשום "לא יודע/ת", ההתייחסות היא לכל הסעיף.**

שאלה 1 (25 נקודות)

מיון חביתיות (pancake sort) הינו בעיית מיון שבו מותרת פעולה אחת בלבד: היפוך רישא של סידרת המספרים הנתונה. דמיינו את המספרים כערימת חביתיות שיש למיין אותן לפי גודלן. זאת אומרת שהפעולה המותרת היא הכנסת מזלג לערימה והיפוך החביתיות שמעל. לדוגמא, בסדרת המספרים המתוארת בצד שמאל (המספר הראשון למעלה), הכנסת מזלג במיקום המתואר והיפוך המספרים שמעליה מביא לסידרה הנתונה מצד ימין.

רכילות: ביל גייטס עבד על נושא תיאורטי אחד לפני שפנה לעושר. הוא עבד על מיון חביתיות.



סעיף א (8 נק'): כתבו פונקציה

```
public static void flip(int fork, int[] array)
```

אשר מקבלת מערך של מספרים `array` ומיקום של מזלג `fork` והופכת את סדר המספרים שבמערך מתחילתו ועד למקום `fork` (כולל). למשל, קטע הקוד הבא:

```
int[] a = {14,12,1,17,6,5,3,3};
flip(3,a);
for (int i=0; i<a.length; i=i+1)
     System.out.print(a[i] + " ");
System.out.println();
```

ידפיס את השורה הבאה:

```
17 1 12 14 6 5 3 3
```

ניתן להניח שהמערך `array` אינו `null` ואינו ריק וגם ש- `fork` הינו אינדקס חוקי במערך. מותר להוסיף משתנים מטיפוס `int` בלבד. אין להגדיר משתנים אחרים. אין להשתמש בפונקציות עזר.

סעיף ב (17 נק'): כתבו פונקציה

```
public static void pancakeSort(int[] array)
```

אשר מקבלת מערך של מספרים וממיינת אותו בסדר עולה. מותרת רק פעולה אחת אשר משנה את איברי המערך וזו פעולת ה `flip` מסעיף א'. מותר להשתמש בה גם אם לא הגדרתם אותה. ניתן להניח שהמערך `array` אינו `null`. מותר להוסיף משתנים מטיפוס `int` בלבד. אין להגדיר משתנים אחרים. מותר להגדיר ולהשתמש בפונקציית עזר אחת. ניתן להעביר לה ארגומנט מכל סוג אבל מותר להגדיר בה משתנה עזר רק מסוג `int`.

בסעיף זה פתרון יעיל ייחשב כפתרון שממיין `n` מספרים בסדר גודל של n^2 פעולות לכל היותר.

שאלה 2 (25 נקודות):

נתון הממשק Comparable אותו ראינו בכיתה.

```
public interface Comparable {
/**
* Decides the order between this object and
* another given object
* @param other an object to compare with this
* Comparable object
* @return a negative value, 0, or a positive value,
* if this object has a lower, an identical, or a
* higher rank in the order, respectively.
*/
     public int compareTo(Object other);
}
```

סעיף א' (10 נקודות): כתבו פונקציה

```
public static Comparable getMax(Comparable[] arr)
```

אשר מקבלת מערך עצמים מטיפוס Comparable ומחזירה את האובייקט הגדול ביותר על פי הסדר המוגדר על ידי השיטה compareTo (האובייקט הגדול ביותר הינו איבר שאין גדול ממנו).

הנחות: ניתן להניח שהקלט אינו null ושגודל המערך גדול מ-0.
ניתן להניח שקיים במערך הקלט **בדיוק אובייקט אחד** גדול ביותר.

סעיף ב' (15 נקודות):

נתונה המחלקה **המופשטת** Person

```
public abstract class Person implements Comparable {
     protected String name;

     public Person(String name) {
          this.name = name;
     }
     public String toString(){return name;}
}
```

- שדה המחלקה היחיד הוא name, שמו של העצם.
- הבנאי היחיד של המחלקה מאתחל את שם העצם.
- שיטת toString() מחזירה מחרוזת שהינה שמו של העצם.

עליכם להשלים בדף התשובות את המחלקה Student. מחלקה זו מרחיבה את המחלקה האבסטרקטית Person.

עליכם להשלים את בנאי המחלקה כך שעצם מטיפוס Student יתואר על ידי שם (name) וציון (grade). כמו כן עליכם להשלים את השיטה compareTo המממשת השוואה בין שני עצמים מטיפוס Student. ההשוואה בין הסטודנטים תתבצע ע"פ הציון שלהם (כלומר אנו נגדיר שסטודנט א' גדול מסטודנט ב' ביחס המוגדר אם הציון של סטודנט א' גדול מהציון של סטודנט ב'). במידה ולשני הסטודנטים ציון זהה, ההשוואה תתבצע ע"פ השוואה לקסיקוגרפית (כלומר הסטודנט הגדול מבינהם הוא זה שמופיע קודם בסדר המילוני).

תזכורת: המחלקה String מממשת את הממשק Comparable והשוואת מחרוזות מתבצעת לקסיקוגרפית.

ניתן להניח שאובייקט הקלט של השיטה compareTo הינו תקין, כלומר מטיפוס המחלקה Student ואינו null.

```
public class Student extends Person{

     protected double grade;

     public Student(String name, double grade){
          //השלמו בדף התשובות
     }

     public int compareTo(Object other){
          //השלמו בדף התשובות
     }
}
```

להלן דוגמאת הרצה:

```
public static void main(String[] args){
          Student s1 = new Student("Kermit",88);
          Student s2 = new Student("Bentz",80);
          Student s3 = new Student("Arik",88);
          Student[] arr1 = new Student[2];
          arr1[0] = s1;
          arr1[1] = s2;
          System.out.println(getMax(arr1));
          //Output: Kermit

          Student[] arr2 = new Student[3];
          arr2[0] = s1;
          arr2[1] = s2;
          arr2[2] = s3;
          System.out.println(getMax(arr2));
          //Output: Arik
}
```

שאלה 3 (25 נקודות)

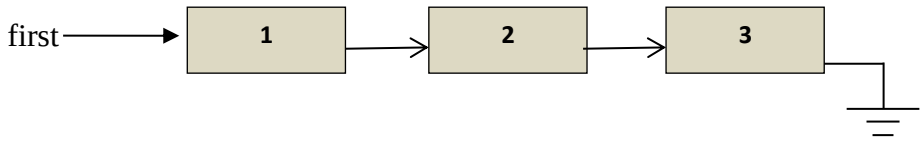
בשאלה זו נתייחס לרשימות מקושרות שבחוליות שלהן יש ערכים שלמים **וחיוביים**.
נתונה לכם המחלקה IntLink הבאה המתארת חוליה ברשימה המקושרת:

```
public class IntLink {

      public int data;
      public IntLink next;

      public IntLink(int data, IntLink next) {
            this.data = data;
            this.next = next;
      }
      public IntLink(int data) {
            this(data, null);
      }
}
```

שימו לב: בשאלה זו נגדיר רשימה מקושרת ע"י מצביע מטיפוס IntLink לחוליה הראשונה ברשימה (המצביע first בדוגמא הבאה).

לדוגמא, הקוד הבא יוצר את הרשימה המקושרת שבתמונה:

```
IntLink third  = new IntLink(3);
IntLink second = new IntLink(2,third);
IntLink first  = new IntLink(1,second);
```



סעיף א' (12 נק')

נתון הממשק הבא:

```
public interface Filter {
      public boolean accept(Object obj);
}
```

מחלקה המממשת את הממשק Filter מאפשרת להבחין בין עצמים לפי קריטריון שבחרנו מראש.
לדוגמא: המחלקה IntLinkFilter המממשת את הממשק Filter מאפשרת להבחין בין עצמים מטיפוס IntLink לבין כל שאר העצמים.

```
public class IntLinkFilter implements Filter {
      public boolean accept(Object obj) {
            return (obj instanceof IntLink);
      }
}
```

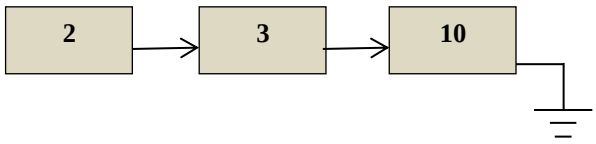
עליכם להשלים בדף התשובות את המחלקה HierarchyFilter המממשת את הממשק Filter ומאפשרת להבחין בין רשימות מקושרות **היררכיות** מטיפוס IntLink לבין כל עצם אחר.
רשימה נקראת **רשימה היררכית**, אם עבור כל חוליה a, הערך של החוליה a (כלומר a.data), גדול **<u>מסכום כל הערכים</u>** של החוליות שקודמות לחוליה a (כלומר סכום הערכים מהחוליה הראשונה ברשימה עד החוליה a, לא כולל).

ניתן להניח כי:

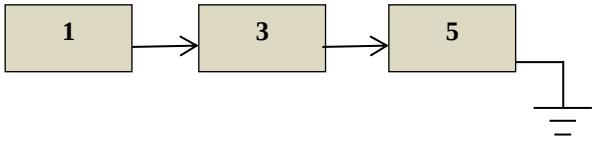
- הקלט המתקבל אינו null.
- הקלט המתקבל הינו אובייקט מטיפוס IntLink המצביע לחוליה הראשונה ברשימה (כמו המשתנה first מהדוגמא בעמוד הקודם) וכן כל החוליות ברשימה הינן מטיפוס IntLink.
- הרשימה תקינה וללא מעגלים.
- רשימה באורך 1 **תמיד** מקיימת את הקריטריון של הפילטר.

```
public class HierarchyFilter implements Filter {
     public boolean accept(Object obj){
          //השלמו בדף התשובות
     }
}
```

שתי דוגמאות לרשימות שעומדות בקריטריון:

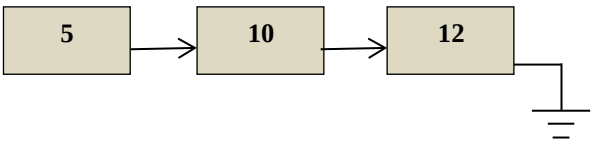


- הערך של החוליה הראשונה (2) גדול מסכום הערכים של החוליות שקודמות לה (0).
- הערך של החוליה השנייה (3) גדול מסכום הערכים של החוליות שקודמות לה (2).
- הערך של החוליה השלישית (10) גדול מסכום הערכים של החוליות שקודמות לה (3+2=5).

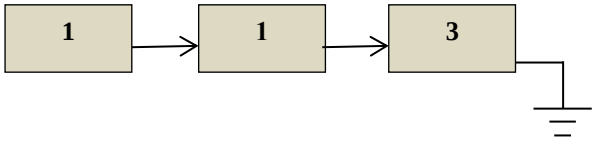


- הערך של החוליה הראשונה (1) גדול מסכום הערכים שקודמות לה (0).
- הערך של החוליה השנייה (3) גדול מסכום הערכים של החוליות שקודמות לה (1).
- הערך של החוליה השלישית (5) גדול מסכום הערכים של החוליות שקודמות לה (3+1=4).

שתי דוגמאות לרשימות **שלא** עומדות בקריטריון:



- הערך של החוליה השלישית (12) **אינו** גדול מסכום הערכים של החוליות שקודמות לה (10+5=15).



- הערך של החוליה השניה (1) **אינו** גדול מסכום הערכים של החוליות שקודמות לה (1).

<u>סעיף ב' (13 נק')</u>

נתונים הממשקים Iterator ו-Iterable אותם ראינו בכיתה:

```
public interface Iterator{
     // returns true if the iteration has more elements
     public boolean hasNext();
     // returns the next element in the iteration
     // throws NoSuchElementException if has no more
     // elements
     public Object next();
}

public interface Iterable {
    // returns an iterator over a set of elements
    public Iterator iterator();
}
```

כמו כן, נתונה המחלקה Vector המגדירה מערך דינמי של אובייקטים מטיפוס Object. המחלקה Vector מממשת את הממשק Iterable. בסעיף זה לא תצטרכו פרטים נוספים על המחלקה Vector לצורך מתן תשובה.

עליכם להשלים בדף התשובות את המחלקה VectorOfIntLinkIterator. מחלקה זו תממש איטרטור עבור Vector של רשימות מקושרות ותאפשר מעבר איטרטיבי אך ורק על הרשימות במבנה הנתונים Vector המקיימות את הקריטריון המוגדר באובייקט מטיפוס Filter.

עליכם להשלים את בנאי המחלקה המקבל Vector של רשימות מקושרות וכן אובייקט מטיפוס Filter. כמו כן עליכם להשלים את השיטה hasNext כך שתחזיר true במידה וקיימת רשימה מקושרת נוספת ב-Vector המקיימת את הקריטריון שהוגדר באובייקט מטיפוס Filter. כמו כן יש להשלים את השיטה next כך שתחזיר (במידה וקיימת) את הרשימה הבאה ב- Vector המקיימת את הקריטריון שהוגדר באובייקט מטיפוס Filter או תזרוק שגיאת NoSuchElementException במידה ואין.

```
public class VectorOfIntLinkIterator implements Iterator{
     private Vector vec;
     private Filter filter;
     private Object nextObj;
     private Iterator iterator;

     public VectorOfIntLinkIterator(Vector vec, Filter
     filter){
          //השלמו בדף התשובות
     }

     public boolean hasNext(){
          //השלמו בדף התשובות
     }
     public Object next() {
          //השלמו בדף התשובות
     }
}
```

שאלה 4 (25 נקודות)

בשאלה זו נתמקד במבנה הנתונים מחסנית. הממשק למבנה הנתונים הינו הממשק הבא (שימו לב כי זהו אותו ממשק שראיתם בכיתה בתוספת השיטה (size):

```
/**
 * Interface describing a standard Stack ADT.
 */
public interface Stack {
	/** Add o to the top of this Stack.
	 * @param o The object to be pushed.
	 */
	public void push(Object o);

	/** Remove and return the top element of this Stack.
	 * @return the formerly top element of the stack.
	 * @throws EmptyStackException if trying to pop from an empty
	 * stack.
	 */
	public Object pop();

	 /** Returns whether the stack is empty.
	  * @return true if and only if the stack is empty
	  */
	public boolean isEmpty();

	/**
	 * Returns the number of items in the stack.
	 * @return the number of items in the stack.
	 */
	public int size();
}
```

בנוסף לממשק נתון לשימושכם מימוש של מחסנית מבוססת מערך המממש את הממשק הנ"ל, ודומה למימוש שראיתם בתרגיל בית 4 (בתוספת השיטה (size):

```
import java.util.EmptyStackException;

/**
 * This class implements the Stack interface by using an
 * extendable array.
 */
public class StackAsArray implements Stack {

	/* Defines the initial capacity of a stack */
	private static final int INITIAL_CAPACITY = 5;
	/* Defines the capacity extension of a stack when it is full */
	private static final int CAPACITY_EXTENSION = 5;

	protected Object[] elements;
	protected int size;

	/**
	 * Creates a new stack.
	 */
	public StackAsArray() {
		elements = new Object[INITIAL_CAPACITY];
		size = 0;
	}
```

```
	public void push(Object o) {
		// Make sure there is enough room in the array
		if (size >= elements.length)
			extend();

		elements[size] = o;
		size = size + 1;
	}

	public Object pop() {
		// If there are no elements, an exception is thrown
		if (size == 0)
			throw new EmptyStackException();

		size = size - 1;
		return elements[size];
	}

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

	public int size() {
		return size;
	}

	/**
	 * Extends the size of the array by CAPACITY_EXTENSION.
	 */
	private void extend() {
		// Create a new array with the new size
		Object[] tmpArray = new Object[elements.length +
					CAPACITY_EXTENSION];

		// Copy the old elements to the new array
		for (int i = 0; i < elements.length; i=i+1)
			tmpArray[i] = elements[i];

		// Replace the elements array with the new one
		elements = tmpArray;
	}
}
```

<u>סעיף א (10 נקודות)</u>

בסעיף זה עליכם לממש את המחלקה PeekableStackAsArray, אשר מייצגת את מבנה הנתונים מחסנית-הצצה (מחסנית עם היכולת להציץ בתוכנה). פעולת ההצצה מחזירה את הערך במקום הנתון יחסית לראש המחסנית, מבלי להסיר את הערך מהמחסנית. על המחסנית לתמוך בפעולות הבאות:

- boolean isEmpty() – מחזירה true אם ורק אם אין ערכים במחסנית.
- void push(Object o) – דוחפת את הערך o לראש המחסנית.
- Object pop() – מוציאה את הערך שנמצא בראש המחסנית, ומחזירה אותו. במידה ולא קיימים ערכים במחסנית, על הפעולה לזרוק חריגה מסוג EmptyStackException.
- int size() – מחזירה את מספר הערכים שנמצאים במחסנית.
- Object peek(int i) – מחזירה את הערך הנמצא במקום ה – i יחסית לראש המחסנית, כאשר peek(0) תחזיר את הערך בראש המחסנית, peek(1) תחזיר את הערך שמתחת

לראש המחסנית, וכו'. במידה ומנסים להציץ על ערך שאינו קיים, תיזרק החריגה EmptyStackException.

- void clear() – מנקה את כל איברי המחסנית (מביאה למצב בו המחסנית ריקה).

על המחלקה PeekableStackAsArray להרחיב את המחלקה StackAsArray הנ"ל, ולממש את הממשק PeekableStack:

```
/**
 * Interface describing an extended Stack, with random access through
"peeking".
 */
public interface PeekableStack extends Stack {
	/**
	 * Returns the i-th item from the top of the stack, where the
	 *  top is the 0-th item.
	 * @return the i-th stack item (counting from 0).
	 * @throws EmptyStackException if trying to peek at an
	 *  element which is not in the stack.
	 */
	public Object peek(int i);
	/**
	 * Empties the stack of all its contents.
	 */
	public void clear();
}
```

<u>סעיף ב' (15 נקודות)</u>

בסעיף זה נעסוק במבנה הנתונים מחסנית (כמתואר בסעיף א') ובמחלקות CalcToken, BinaryOp ו- ValueToken כמתואר בתקציר המחלקות הבא:

abstract class CalcToken	
חתימה	**תכונה**
String toString()	החזרת מחרוזת המתארת את תוכן הסמל

abstract class BinaryOp extends CalcToken	
חתימה	**תכונה**
double operate(double left, double right)	החזרת התוצאה המיוצגת ע"י אופרטור בינארי על שני הפרמטרים המועברים. אם הפעולה היא $, אז מוחזרת התוצאה: left $ right

class ValueToken extends CalcToken	
חתימה	**תכונה**
ValueToken(double value)	בנאי המאתחל את הסמל הנוצר שיכיל את הערך המספרי value
double getValue()	החזרת הערך המספרי המיוצג בסמל

כפי שראיתם בעבודה 4, המחלקה CalcToken מייצגת סמל (token) יחיד בביטוי מתמטי, המחלקה BinaryOp מייצגת פעולה בינארית מתמטית (לדוגמה: חיבור וחיסור) בביטוי מתמטי, והמחלקה ValueToken מייצגת ערך מספרי בביטוי מתמטי.

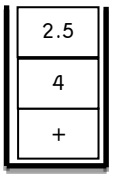
ביטוי מתמטי בצורת postfix הינו ביטוי המוגדר בצורה הרקורסיבית הבאה:

1. כל מספר הוא ביטוי postfix.
2. אם P1 וגם P2 הם ביטויי postfix, ו- Op הינו אופרטור בינארי (פעולה מתמטית על שני ביטויים), אזי "P1 P2 Op" הוא ביטוי postfix.

בסעיף זה עליכם לממש את הפונקציה (הסטטית): double evaluatePostfix(Stack s) המקבלת כקלט מחסנית s (מטיפוס Stack כפי שמתואר בסעיף א'), המכילה את כל הסמלים (tokens) בביטוי postfix חוקי מסוים, ומחזירה את ערך הביטוי לאחר שיערוך. שימו לב, בניגוד לעבודת הבית, כאן ניתן להניח שביטוי ה- postfix המחרוזתי כבר חולק בצורה טובה לסמלים וכל הסמלים הוכנסו למחסנית s. כאשר הסמל בראש המחסנית הוא הסמל השמאלי ביותר בביטוי, והסמל בתחתית המחסנית הוא הסמל הימני ביותר בביטוי. לדוגמה:

הביטוי: "2.5 4 +"

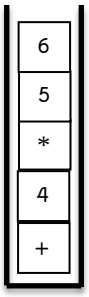
המחסנית:



התוצאה: 2.5 + 4 = 6.5

הביטוי: "6 5 * 4 +"

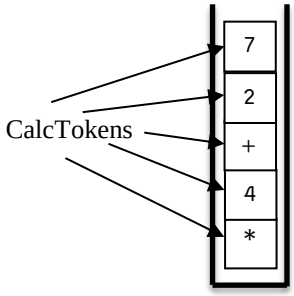
המחסנית:



התוצאה: (6 * 5) + 4 = 34.0

הביטוי: "7 2 + 4 *"

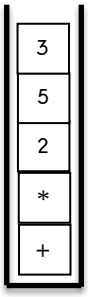
המחסנית:



התוצאה: (7 + 2) * 4 = 36.0

הביטוי: "3 5 2 * +"

המחסנית:



התוצאה: 3 + (5 * 2) = 13.0

ניתן להניח כי המחסנית s איננה null, וכל האובייקטים במחסנית s הינם מטיפוס CalcToken, ובפרט מאחד משני הטיפוסים ValueToken או BinaryOp הנ"ל, וכי הביטוי במחסנית הוא ביטוי postfix חוקי.

רמז: מומלץ להגדיר משתנה נוסף מטיפוס מחסנית ולהשתמש בו לחישוב הביטוי.