מבוא למדעי המחשב
202-1-101-1

מבוא לתכנות למערכות מידע
202-1-104-1

סמסטר א', תשס"ז, 2006/7
בחינת מועד א' – 28.1.2007

ד"ר ג'יהאד אל-סאנע
מר איתי בר-יוסף
גב' דקלה דותן
פרופ' מיכאל קודיש
ד"ר צחי רוזן
מר גיא שני

משך הבחינה: שעתיים וחצי.
חומר עזר: אסור.

בבחינה זו 5 שאלות. ענו על כולן. הניקוד מסתכם ב-100 נקודות.

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. מותר להסתמך על סעיפים קודמים גם אם לא עניתם עליהם. **שימו לב:** בשאלות התכנות החשיבות העליונה היא לנכונות הקוד, מאידך, יעילות וסגנון חשובים גם הם, ולכן תשובה יעילה ומסוגננת תזכה בציון גבוה יותר.

רשמו את תשובותיכם במקומות המיועדים לכך בטופס הבחינה בלבד. המחברת שקיבלתם היא מחברת טיוטה ולא תימסר כלל לבדיקה.

בהצלחה!

שאלה 1 (20 נקודות)

בשאלה זו שני סעיפים. ניתן להשתמש בשיטה מסעיף א' עבור סעיף ב', גם אם לא הצלחתם לממשה.

סעיף א (10 נקודות)

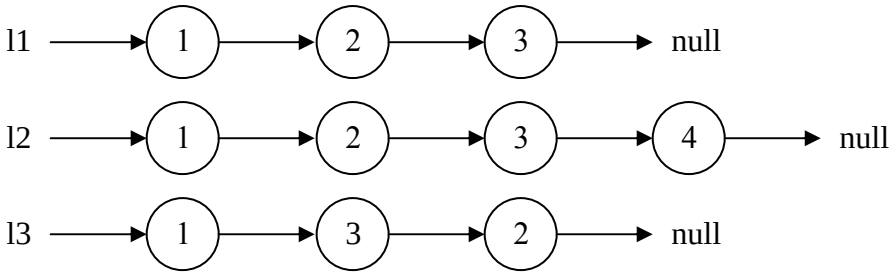
נתונה המחלקה Link:

```
public class Link{
        private Link m_lNext;
        private Object m_aData;
        <constructors>
        <accessors>
}
```

נאמר כי הרשימה list1 הינה תחילית (prefix) של הרשימה list2 אם אורכה של list1 קטן או שווה מאורכה של list2 ולכל i המידע באיבר ה-i ב-list1 זהה לוגית למידע באיבר ה-i ב-list2.

הוסיפו **למחלקה Link** שיטה **רקורסיבית** בשם isPrefixOf המקבלת פרמטר מטיפוס Link ובודקת האם הרשימה המתחילה בעצם המפתח הינה תחילית (prefix) של הרשימה המתחילה בפרמטר.

לדוגמא – בהינתן המשתנים l1, l2, l3 מטיפוס Link:




```
System.out.println( l1.isPrefixOf( l2 ) ); //prints true
System.out.println( l2.isPrefixOf( l1 ) ); //prints false
System.out.println( l3.isPrefixOf( l2 ) ); //prints false

public boolean isPrefixOf( Link lOther ) {

        if ( lOther == null )
                return false;

        if ( ! getData().equals( lOther.getData() ))
                return false;

        if ( ( getNext() == null ) )
                return true;

        return getNext().isPrefixOf( lOther.getNext() );

}
```

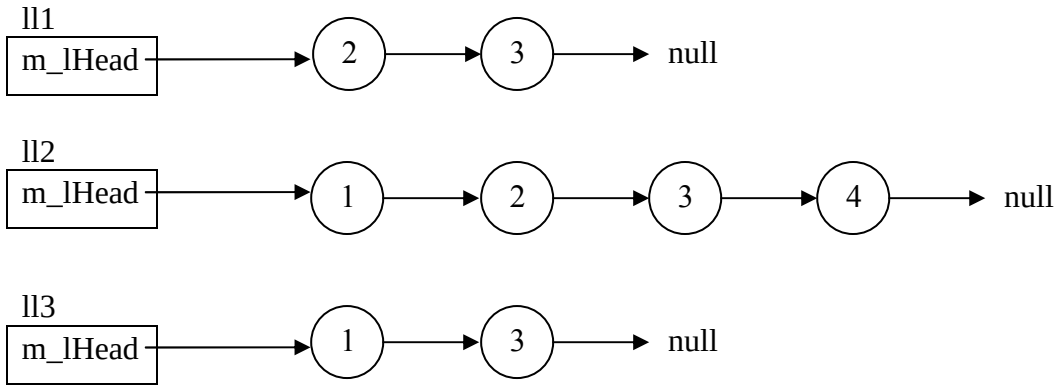
סעיף ב (10 נקודות)

נתונה המחלקה LinkedList להלן. **אין להשתמש בשיטות אחרות מהמחלקה.**

```
public class LinkedList{
        private Link m_lHead;
        public LinkedList() { m_lHead = null; }
        public void addHead( Object oData ) {…}
        public void addTail( Object oData ) {…}
        public boolean contains( Object oData ) {…}
        public Object removeHead() {…}
        public String toString() {...}
}
```

הוסיפו למחלקה שיטה בשם isSublistOf המקבלת פרמטר מטיפוס LinkedList ובודקת האם עצם המפתח הינו תת-רשימה (רציפה) של הפרמטר. אין לשנות את הרשימות.

לדוגמא, הרשימה (2, 3, 4) היא תת-רשימה של (1, 2, 3, 4, 5, 6), אך (3, 4, 6) אינה תת-רשימה של אף אחת מהרשימות הנ"ל. דוגמא נוספת – בהינתן המשתנים ll1, ll2, ll3 מטיפוס LinkedList:




```
System.out.println( ll1.isSublistOf( ll2 ) ); //prints true
System.out.println( ll2.isSublistOf( ll1 ) ); //prints false
System.out.println( ll3.isSublistOf( ll2 ) ); //prints false

public boolean isSublistOf( LinkedList llOther ) {
        if ( m_lHead == null )
                return true;

        if ( llOther == null )
                return false;

        Link lOtherLink = llOther.m_lHead;
        while ( lOtherLink != null ) {
                if ( m_lHead.isPrefixOf(lOtherLink) )
                        return true;
                lOtherLink = lOtherLink.getNext();
        }

        return false;

}
```

שאלה 2 (20 נקודות)

בשאלה זו אין קשר בין הסעיפים. יש לממש את המחלקות והשיטות באופן פשוט ויעיל. מימושים מסורבלים יזכו לניקוד חלקי בלבד.
בכיתה ראינו מימושים של הממשק Iterator אשר עוברים באופן סידרתי על איברי מבני נתונים כגון רשימה משורשרת וקבוצה. איטרטורים יכולים גם לעבור באופן סידרתי על אוספי איברים אינסופיים, כמו המספרים הטבעיים.

סעיף א (5 נקודות)

כתבו את המחלקה NaturalNumbersIterator המממשת את ממשק Iterator ועוברת באופן שיטתי על המספרים הטבעיים. כלומר, בכל קריאה לשיטה next() יוחזר מופע (instance) של המחלקה Integer אשר מייצג את המספר הטבעי הבא. התעלמו מהמגבלה הטכנית לפיה ישנו גבול עליון עבור מספרים שלמים ב-Java.
לדוגמא:

```
Iterator itNaturals = new NaturalNumbersIterator();
for( int i = 0 ; i < 10 ; i = i + 1 )
        System.out.print(itNaturals.next() + "," );
```

קטע קוד זה ידפיס: 1,2,3,4,5,6,7,8,9,10,

```
public class NaturalNumbersIterator implements Iterator {

        private int m_iNext;

        public NaturalNumbersIterator() {
                m_iNext = 1;
        }

        public boolean hasNext() {
                return true;
        }
        public Object next() {

                Integer iAns = new Integer( m_iNext );
                m_iNext = m_iNext + 1;

                return iAns;
        }

} // class NaturalNumbersIterator
```

לעזרתכם הגדרה חלקית של המחלקה Integer:

```
public class Integer{
        private int m_iValue;
        public Integer( int iValue ){ m_iValue = iValue; }
        public int intValue(){ return m_iValue; }
        public String toString(){ return m_iValue  + ""; }
}
```

סעיף ב (5 נקודות)

סדרת פיבונאצ'י מוגדרת באופן הבא – $fib_0=fib_1=1$, ועבור כל $n>1$: $fib_n=fib_{n-1}+fib_{n-2}$.

כתבו את המחלקה FibonacciIterator המממשת את ממשק Iterator ועובדת באופן שיטתי על מספרי פיבונאצ'י. כלומר, בכל קריאה לשיטה next() יוחזר מופע (instance) של המחלקה Integer אשר מייצג את מספר פיבונאצ'י הבא. לדוגמא:

```
Iterator itFibs = new FibonacciIterator();
for( int i = 0 ; i < 10 ; i = i + 1 )
        System.out. print( itFibs.next() + "," );
```

קטע קוד זה ידפיס: 1,1,2,3,5,8,13,21,34,55,

```
public class FibonacciIterator implements Iterator {

        private int m_iCurrent;
        private int m_iNext;

        public FibonacciIterator() {

                m_iCurrent = 0;
                m_iNext = 1;
        }

        public boolean hasNext() {

                return true;
        }

        public Object next() {

                Integer iAns = new Integer( m_iNext );
                int iTmp = m_iCurrent + m_iNext;
                m_iCurrent = m_iNext;
                m_iNext = iTmp;
                return iAns;
        }

} // class FibonacciIterator
```

סעיף ג (10 נקודות)

הממשק Filter מאפשר לנו לסנן איברים של מבנה נתונים:

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

כתבו את המחלקה OnlyOnceFilter המממשת את ממשק Filter כך שתייצג פילטר המקבל (בשיטה accept) אך ורק איברים "חדשים", כלומר איברים שאינם זהים לוגית לאיברים שהתקבלו ע"י הפילטר קודם לכן. לדוגמא, בהינתן המחלקה LinkedList:

```
public class LinkedList {
	private Link m_lHead;
	public LinkedList() { m_lHead = null; }
	public void addHead( Object oData ) {…}
	public void addTail( Object oData ) {…}
	public boolean contains( Object oData ) {…}
	public Object removeHead() {…}
	public String toString() {...} // מחזירה מחרוזת של איברי הרשימה מופרדים ע"י פסיקים
	public LinkedList filter( Filter f ) {...}
}
```

קטע הקוד הבא:

```
LinkedList l1 = new LinkedList();
for( int i = 0 ; i < 10 ; i = i + 1 )
	l1.addTail( new Integer( i % 3 ) );
LinkedList l2 = l1.filter( new OnlyOnceFilter() );
System.out.println( l1 );
System.out.println( l2 );
```

ידפיס:

```
(0, 1, 2, 0, 1, 2, 0, 1, 2, 0)
(0, 1, 2)
```

הערה - בסעיף זה ניתן להשתמש בכל מבנה נתונים שנלמד בכיתה מבלי לממשו.

```
public class OnlyOnceFilter implements Filter {

	private Set m_sSeenObjects;

	public OnlyOnceFilter() {

		m_sSeenObjects = new LinkedListSet();

	}
	public boolean accept( Object obj ) {

		if ( m_sSeenObjects.contains( obj ) )
			return false;
		m_sSeenObjects.add( obj );
		return true;

	}
} // class OnlyOnceFilter
```

שאלה 3 (20 נקודות)

נתונות המחלקות Point, ConvexPolygon (כפי שהוגדרו בתרגיל 4) ו-Triangle.
המחלקה ConvexPolygon מייצגת מצולע קמור בעזרת מערך של קודקודים, כאשר ישנה צלע בין כל שני קודקודים סמוכים במערך וכן בין הקודקוד הראשון והקודקוד האחרון. ניתן להניח כי בכל מצולע קמור ישנן לפחות 3 נקודות.
המחלקה Triangle מייצגת משולש בעזרת שלושה קודקודים, כאשר בין כל שניים ישנה צלע.

```
public class Point {
        private double m_dX, m_dY;
        public Point( double dX, double dY ){…}
        public double distance( Point pOther ){…}
        public void translate( double dDeltaX, double dDeltaY ) {...}
}
public class ConvexPolygon {
        private Point[] m_apPoints;
        public ConvexPolygon( Point[] appoints ){…}
        public void translate( double dDeltaX, double dDeltaY ) {...}
}
public class Triangle{
        private Point m_p1, m_p2, m_p3;
        public Triangle( Point p1, Point p2, Point p3 ){…}
        public double area(){…} // מחזירה את שטח המשולש
        public double perimeter(){…} // מחזירה את היקף המשולש
        public void translate( double dDeltaX, double dDeltaY ) {...}
}
```

סעיף א (10 נקודות)

הוסיפו למחלקה ConvexPolygon את השיטה perimeter() המחשבת את היקף המצולע.

```
public double perimeter() {

        double dAns = 0.0;
        for (int i = 0; i < m_apPoints.length; i = i + 1)
                dAns = dAns + m_apPoints[i].distance(m_apPoints[(i+1)%m_apPoints.length]);
        return dAns;

}
```

סעיף ב (10 נקודות)

הוסיפו למחלקה ConvexPolygon את השיטה area() המחשבת את שטח המצולע.
רמז – היעזרו בשיטת השטח של המחלקה Triangle המצורפת.

```
public double area() {

        double dAns = 0.0;
        Triangle triangle = null;
        for (int i = 1; i < m_apPoints.length-1; i = i+1) {
                triangle = new Triangle(m_apPoints[0], m_apPoints[i], m_apPoints[i+1]);
                dAns = dAns + triangle.area();
        }
        return dAns;

}
```

שאלה 4 (20 נקודות)

קראו בעיון את ההגדרות בשאלה זו. לא יינתן ניקוד עבור פתרון שאינו תואם במדויק להגדרה של חלוקה כמעט מאוזנת.

נאמר כי חלוקה של מערך מספרים הינה **כמעט מאוזנת** אם ניתן לחלק את המספרים במערך לשתי קבוצות כאשר:

- סכום המשקלות בשתי הקבוצות זהה.
- באחת הקבוצות יש **לכל היותר** שני איברים יותר מאשר בקבוצה השנייה.

השלימו את הגדרת הפונקציה boolean balancedPartition(int[] array) המקבלת מערך array של משקולות (ערכים שלמים חיוביים) ומחזירה ערך true אם ורק אם קיימת חלוקה כמעט מאוזנת של המערך array.
לדוגמא, אם array = {1,2,2,3,5,6,1}, הקריאה לפונקציה תחזיר ערך true כיון שניתן לחלק את המשקולות לקבוצות {2,2,6} ו- {1,1,3,5} בהן סכום המשקלות זהה ובקבוצה אחת יש איבר אחד יותר מאשר בקבוצה השנייה.
מנגד, אם array = {1,2,3,5,6}, הקריאה לפונקציה תחזיר ערך false כיון שאין חלוקה של כל המשקולות לשתי קבוצות שוות משקל.
אם array = {1,2,3,5,6,17}, הקריאה לפונקציה תחזיר ערך false, כיון שלא קיימת חלוקה בה הקבוצות הן שוות משקל וכמעט מאוזנות. בחלוקה לקבוצות {17} ו- {1,2,3,5,6} סכום המשקלות אמנם זהה, אך חלוקה זו אינה כמעט מאוזנת, שכן ההפרש בין מספרי האיברים בקבוצות הינו 4.

הניחו כי המערך array אינו null ושכל הערכים בו שלמים וחיוביים. שימו לב כי כל משקולת חייבת להיות באחת משתי הקבוצות ויכולה להיות רק בקבוצה אחת. שימו לב, אם משקל מופיע מספר פעמים כקלט, עליו להופיע אותו מספר פעמים בחלוקה (למשל המשקל 1 והמשקל 2 בדוגמא הראשונה).

```
public static boolean balancedPartition( int[] array ) {
        return balancedPartition ( arr, 0, 0, 0 _____ );
}

private static boolean balancedPartition ( int[] arr, int index, int diffSum, int diffNum  _____
                                                    _____ ) {
        if ( (index >= arr.length)  &  (diffSum == 0) & (Math.abs(diffNum) <= 2) )  _____
                _____ )
                return true;
        else if (index >= arr.length  _____
                _____ )
                return false;
        else
                return  balancedPartition( arr, index + 1, diffSum + arr[ index ], diffNum + 1 ) |
                        balancedPartition( arr, index + 1, diffSum - arr[ index ], diffNum - 1 ) ;
}
```

שאלה 5 (20 נקודות)

התבוננו במחלקות ובממשקים הנתונים בעמוד האחרון וענו על הסעיפים הבאים.

סעיף א (15 נקודות)

רשמו לצד כל שורה המדפיסה למסך את הפלט המתקבל.

```
A a = new A( 1 );
B b = new B( 1 );
C c = new C( 1 );
D d = new D( 1 );
I i = new C( 1 );
J j = new B( 1 );
System.out.println( "1) " + a.f() );
System.out.println( "2) " + a.g() );
System.out.println( "3) " + a.h() );
System.out.println( "4) " + b.g() );
System.out.println( "5) " + b.h() );
System.out.println( "6) " + c.g() );
System.out.println( "7) " + c.h() );
System.out.println( "8) " + d.h() );
System.out.println( "9) " + i.f() );
System.out.println( "10) " + j.g() );
a = b;
System.out.println( "11) " + a.f().equals( b.f() ) );
System.out.println( "12) " + ( a == b ) );
a = d;
System.out.println( "13) " + a.g() );
b = (B)a;
System.out.println( "14) " + b.g().equals( a.g() ) );
System.out.println( "15) " + ( a == b ) );
```

1) `A.f 1` _____

2) `A.g 1` _____

3) `A.h 2` _____

4) `B.g A.h 4` _____

5) `A.h 5` _____

6) `B.g C.h B.g C.h A.h 7` _____

7) `C.h A.h 9` _____

8) `A.h 4` _____

9) `B.f A.f 2` _____

10) `B.g A.h 4` _____

11) `true` _____

12) `true` _____

13) `D.g B.g A.h 6` _____

14) `false` _____

15) `true` _____

סעיף ב (5 נקודות)

נתונות הגדרות המשתנים הבאות:

```
A a = new A( 1 );
B b = new B( 1 );
C c = new C( 1 );
B b1 = new D( 1 );
I i = new C( 1 );
J j = new B( 1 );
```

להלן מספר שורות קוד המבצעות השמות בין המשתנים.

כל שורה עומדת בפני עצמה ואינה תלויה בשורות האחרות.

תקין _____ a = c;

שגיאת קומפילציה, לא ניתן לתקן _____ b = a;

שגיאת קומפילציה, ניתן לתקן _____ c = i; c = (C) i;

שגיאת קומפילציה, לא ניתן לתקן _____ j = a;

שגיאת זמן ריצה_____ c = (C) b1;

רשמו עבור כל שורת קוד בדיוק אחד מהדברים הבאים:

- "תקין", אם השורה לא תגרום לשגיאת קומפילציה ולא תגרום לשגיאת זמן ריצה.
- "שגיאת קומפילציה", אם השורה תגרום לשגיאת קומפילציה.
- אם ניתן לתקן את השורה כך שתעבור קומפילציה ולא תגרום לשגיאת זמן ריצה, כתבו שורה תקינה המבצעת את אותה פעולה כמו השורה השגויה. אם לא ניתן לתקן כך, כתבו "לא ניתן לתקן".
- "שגיאת זמן ריצה", אם השורה לא תגרום לשגיאת קומפילציה אך כן תגרום לשגיאה בזמן ריצה.

ניתן להסיר עמוד זה מהמבחן ולהגיש את המבחן ללא העמוד.
נתונים הממשקים והמחלקות להלן:

```
public interface I {
	public String f();
}
```

```
public interface J {
	public String g();
}
```

```
public class A implements I {
	private int m_i;
	public A( int i ){
		m_i = i;
	}
	public String f(){
		return "A.f " + m_i;
	}
	public String g(){
		return "A.g " + m_i;
	}
	public int increment(){
		m_i = m_i + 1;
		return m_i;
	}
	public String h(){
		m_i = m_i + 1;
		return "A.h " + m_i;
	}
}
```

```
public class B extends A implements J {
	public B( int i ){
		super( i + 1 );
	}
	public String f(){
		return "B.f " + super.f();
	}
	public String g(){
		int x = increment();
		return "B.g " + h();
	}
}
```

```
public class C extends B {
	public C( int i ){
		super( i );
	}
	public String g( int x ){
		return "C.g " + super.g() + x;
	}
	public String h(){
		if( increment() < 5 )
			return "C.h " + g();
		return "C.h " + super.h();
	}
}
```

```
public class D extends B {
	public D( int i ){
		super( i + 1 );
	}
	public String g(){
		return "D.g " + super.g();
	}
	public String h( int x ) {
		return "D.h " + x;
	}
}
```

בהצלחה!