



**שאלה מספר 1 (30 נקודות)**

בשאלה הבאה עליכם לכתוב קוד. ניתן לפתור את השאלה פתרון מלא ע"י מילוי המלבנים הריקים שבקטעי הקוד. אין חובה למלא את כל המלבנים שבתבנית הפתרון.

ברצוננו להוסיף למחלקות קיימות את האפשרות להשוות בין שני עצמים מאותה המחלקה. על ההשוואה לתמוך ב-6 היחסים הבאים:  
 $>$ ,  $<$ ,  $=$ ,  $\neq$ ,  $\leq$ ,  $\geq$

השוואה בין שני עצמים מאותו טיפוס תעשה ע"י הממשק `MyComparable` הנתון להלן:

```
public interface MyComparable {

    boolean lessThanEqual (MyComparable other);

    boolean lessThan (MyComparable other);

    boolean greaterThanEqual (MyComparable other);

    boolean greaterThan (MyComparable other);

    boolean equal (MyComparable other);

    boolean notEqual (MyComparable other);

}
```

מתכנתת הגדירה מחלקה המייצגת נקודות במישור שניתן להשוות ביניהן. קריטריון ההשוואה הוגדר להיות שיעור ה- $x$  של הנקודות ובמקרה שלשתי הנקודות שיעור  $x$  זהה ההשוואה תעשה לפי שיעור ה- $y$ . כדי למנוע את שכפול הקוד הגדירה אותה מתכנתת מחלקה מופשטת (abstract class) המממשת את הממשק `MyComparable` ומפשטת את הגדרת המחלקות הקונקרטיות.

להלן הקוד המלא של המחלקה `MyComparablePoint`:

```
public class MyComparablePoint extends AbstComparable {

    public MyComparablePoint(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public boolean lessThanEqual(MyComparable other) {

        MyComparablePoint otherPoint = (MyComparablePoint)other;

        if (x == otherPoint.x)
            return y <= otherPoint.y;
        return x < otherPoint.x;
    }

    public void translate(int dx, int dy) {
        x += dx;
        y += dy;
    }

    protected int x;
    protected int y;
}
```

א. (10 נקודות) ממשו את המחלקה `AbstComparable` המופיעה בקוד לעיל. על המחלקה להיות כללית מספיק כך שתשמש בסיס לכתיבת מגוון רחב של מחלקות (ולא רק למחלקה `MyComparablePoint`). הימנעו ככל הניתן משכפול קוד:

```
public abstract class AbstComparable implements MyComparable {  
  
    public abstract boolean lessThanEqual (MyComparable other);  
  
    public boolean greaterThan (MyComparable other) {  
        _____  
    }  
  
    public boolean lessThan (MyComparable other) {  
        _____  
    }  
  
    public boolean equal (MyComparable other) {  
        _____  
    }  
  
    public boolean greaterThanEqual (MyComparable other) {  
        _____  
    }  
  
    public boolean notEqual (MyComparable other) {  
        _____  
    }  
  
}
```

ב. (4 נקודות) נרצה להגדיר את המחלקה `MyNormComparablePoint`. המחלקה תהיה זהה למחלקה `MyComparablePoint` מהסעיף הקודם פרט לקריטריון ההשוואה בין הנקודות. נקודה מטיפוס `MyNormComparablePoint` מוגדרת להיות קטנה מנקודה אחרת (מאותו טיפוס) אם **הנורמה שלה** קטנה מהנורמה של הנקודה האחרת. (הנורמה של  $(x, y)$  מוגדרת להיות  $\sqrt{x^2 + y^2}$ ).

ממשו את המחלקה `MyNormComparablePoint`. שימו לב – על המחלקה לתמוך בכל השרותים שאותם סיפקה `MyComparablePoint`.

```

    _____ MyNormComparablePoint
    _____ {

    public MyNormComparablePoint (int x, int y) {
    _____
    }

    _____

    }
    
```

ג. (6 נקודות) למימוש המחלקה `MyNormComparablePoint` בעזרת ירושה חסרונות ויתרונות. מהו החסרון המרכזי של מימוש זה ומהו היתרון המרכזי שלו?

חסרון:

---



---



---

יתרון:

---



---



---

ד. (10 נקודות) הציעו מימוש חלופי למחלקות `MyComparablePoint` ו-`MyNormComparablePoint` שיפתור את הבעייתיות של שיוצרת הירושה במקרה זה. אין צורך לספק את קוד המחלקות אלא רק לתאר את העיצוב החלופי. יש לספק תאור מילולי וכן תרשים מחלקות (מלבנים וחיצים) לעיצוב החדש. אין צורך לציין בתרשים פרטים שאינם מהותיים לעיצוב החדש.

עיצוב חלופי:

---

---

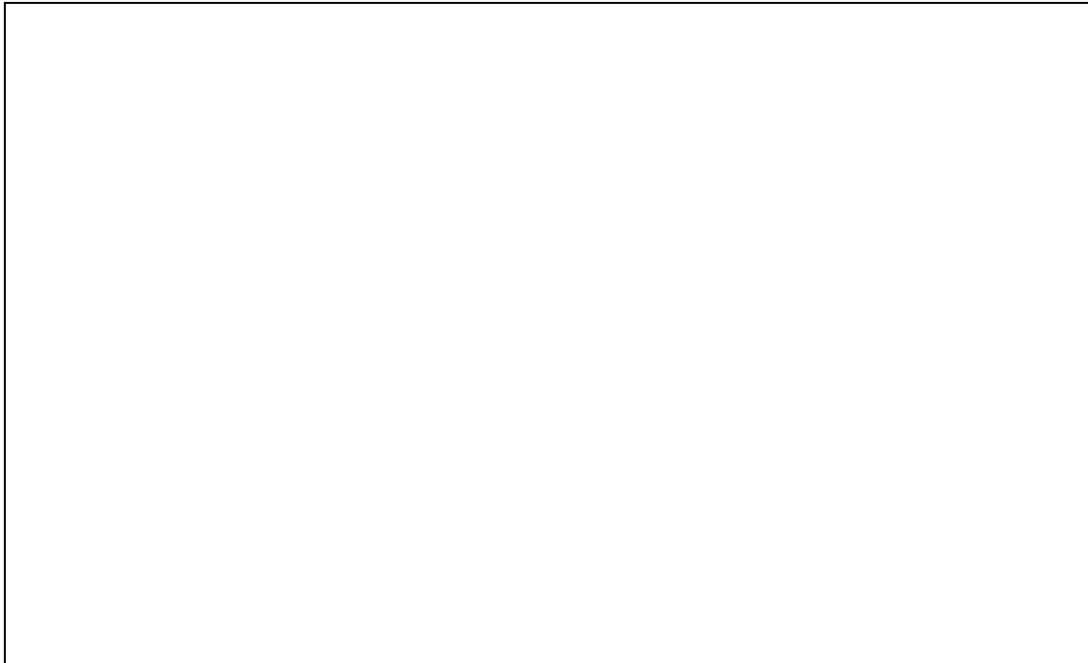
---

---

---

---

תרשים מחלקות:



**שאלה מספר 2 (36 נקודות)**

נתון המנשק הבא עבור רשימות בעלות גודל קבוע שאבריהן שלמים:

```
public interface FixedList {

    public void start();
    // make the first element the current element

    public void forward();
    // assumes the current element is not the last.
    // make the next element the current element

    public boolean atLast();
    // return true iff the current element is the last element

    public void set(int x);
    // change the value of the current element to be x

    public int get();
    // return the value of the current element
}
```

אלה רשימות עם גישה סדרתית בלבד, כלומר קיים בכל רגע אבר נוכחי שאליו מתייחסות הפקודות `get()` ו `set(x)`. ניתן להתקדם לאבר הבא בעזרת `forward()` אם לא נמצאים באבר האחרון. ניתן לעבור לאבר הראשון בעזרת `start()` ו `atLast()` מחזיר `true` אם ורק אם האבר הנוכחי הוא האחרון.

להלן דוגמת שימוש במנשק. `SomeFixedList` היא מחלקה כלשהי שמממשת את המנשק. הבנאי שלה יוצר רשימה שגודלה כערך הפרמטר וכל אבריה הם 0. כתוצאה מהביצוע יודפס

5 0 4

```
FixedList myList = new SomeFixedList(3);
myList.set(5);
myList.forward();
myList.forward();
myList.set(4);
for (myList.start(); !myList.atLast(); myList.forward())
    System.out.print(myList.get() + " ");
System.out.println(myList.get());
```

א. (8 נקודות) כיתבו את החוזה של המנשק; לכל שרות כיתבו תנאי קדם (precondition) ותנאי אחר (postcondition) באופן המקובל (ביטויים בוליאניים שמשמשים בשאילתות). בנוסף, הוסיפו במילים תנאים שלא ניתנים לביטוי בצורה הרגילה.

**public void start( );**

**public void forward( );**

**public boolean atLast( );**

**public void set(int x);**

**public int get( );**



המחלקה הבאה אמורה לממש את המנשק בצורה פשוטה ע"י שימוש במערך לייצוג אברי הרשימה. הגוף של כל הפקודות הושמט (אבל הבנאי נתון).

```
public class SimpleFixedList implements FixedList {
    protected int size;
    protected int current;
    protected int[ ] cont;

    public SimpleFixedList(int n) {
        size = n;
        cont = new int[n];
    }

    public void start() {
        // body omitted
    }

    public void forward() {
        // body omitted
    }

    public boolean atLast() {
        // body omitted
    }

    public void set(int x) {
        // body omitted
    }

    public int get() {
        // body omitted
    }
}
```

ב. (5 נקודות) הגדירו את המשתמר (representation invariant) של המחלקה.

---



---



---



---

ג. (8 נקודות) כתבו את הגוף של כל הפקודות

```

public void start() {
    
}

public void forward() {
    
}

public boolean atLast() {
    
}

public void set(int x) {
    
}

public int get() {
    
}
    
```

ד. (5 נקודות) מוצע לכתוב מחלקה נוספת שמממשת את המנשק, ומשנה את תנאי הקדם של `get()` כך שיבטיח שהאיבר הנוכחי קיבל ערך (במחלקה זאת ערכם של האברים ברשימה בזמן יצירתה אינו מוגדר).

```
// assumes defined()
```

כש `defined()` הוא שאילתה נוספת (שמחזירה `true` אם ורק אם האיבר הנוכחי כבר קיבל ערך). האם זה תקין מבחינת הכללים של תיכון בעזרת חוזה? אם לא, מדוע. אם כן, איך צריך לשנות את תנאי האחר של פעולה זאת (אם זה נדרש).

---



---



---

ה. (5 נקודות) מוצע לכתוב מחלקה נוספת שמממשת את המנשק, ומבטלת את תנאי הקדם של `forward()`. האם זה תקין מבחינת הכללים של תיכון בעזרת חוזה? אם לא, מדוע. אם כן, איך צריך לשנות את תנאי האחר של פעולה זאת (אם נדרש).

---



---



---

ו. (5 נקודות) מוצע לכתוב מחלקה נוספת `ExtendedFixedList` שיוורשת מ `SimpleFixedList` ומוסיפה שאילתא `public int get(int i)` שמחזירה את ערכו של האבר ה-`i` (כאשר `1` מציינ את האבר הראשון, `2` את השני וכן הלאה). תנאי הקדם של השאילתא צריך להיות שאכן קיים אבר `i`. כיתבו את המחלקה `ExtendedFixedList` בשלמותה באופן שתקיים את הכללים של תיכון בעזרת חוזה.

```
public class ExtendedFixedList extends SimpleFixedList {
```

```
}
```

**שאלה מספר 3 (34 נקודות)**  
 א. (12 נקודות) נתונה המחלקה הבאה:

```
public class A {
    public int foo(int i) {
        return i;
    }
}
```

עבור כל אחת מהפונקציות הבאות בדקו האם ניתן להגדירה במחלקה A וסמנו את האפשרות המתאימה: האם תהיה שגיאת קומפילציה (אם כן ציינו מה השגיאה), או שתהיה תעופה בזמן ריצה (אם כן ציינו מאיזה סיבה) או שהקוד או שהקוד ירוץ בצורה תקינה. נמקו בקצרה.

1.

```
public int foo() {
    return 1;
}
```

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין

---



---

2.

```
public int foo (float i) {
    return i;
}
```

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין

---



---

3.

```
public float foo(float i) {
    return i;
}
```

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין

---



---

4.

```
public float foo(int i) {
    return i;
}
```

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין

---



---

ב. (12 נקודות) נתונות המחלקות הבאות:

```

public class A {
    public void foo() {
        System.out.println("in A.foo");
    }

    public static void soo() {
        System.out.println("in A.soo");
    }

    public void moo1() {
        soo();
    }
}

public class B extends A {
    public void foo() {
        System.out.println("in B.foo");
    }

    public static void soo() {
        System.out.println("in B.soo");
    }

    public void moo2() {
        soo();
    }
}

```

ונתון קטע הקוד הבא:

```

public static void main(String[] args) {
    B b = new B();
    A a = b;

    // code here
}

```

עבור כל אחד מהביטויים הבאים, בדקו האם ניתן להגדירו במקום שורת ההערה וסמנו את האפשרות הנכונה: תהיה שגיאת קומפילציה (אם כן ציינו מה השגיאה), התכנית תעוף בגלל חריג שלא טופל (אם כן ציינו מאיזה סיבה), או שהקוד ירוץ בצורה תקינה (אם כן, מה הפלט שיודפס). נמקו בקצרה.

1. a.foo();

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין

---

2. a.moo1();

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין

---

3. a.moo2();

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין

---



---

4. b.foo();

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין

---



---

5. b.moo1();

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין

---



---

6. b.moo2();

שגיאת קומפילציה (מהי?) / תעופה בזמן ריצה (מה הסיבה) / קוד תקין

---



---

ג. (10 נקודות) נתונות המחלקות והממשקים הבאים:

```
public interface I1 {
    public void bar1() throws IOException ;
}
```

```
public interface I2 extends I1 {
    public void bar2() throws Exception;
}
```

```
public interface I3 {
    public void bar3() throws Exception;
}
```

```
public abstract class A implements I2 {
    public void bar2() throws Exception{};
    public void bar3() throws Exception{};
    protected abstract void bar4();
    protected void bar5() {};
}
```

הציעו מימוש למחלקה B עם מינימום שירותים (מתודות) (והשירותים פשוטים ככל האפשר) כך שכל הקוד יהיה תקין:

```
public static class B extends A implements I3 {
```

```
}
```