תכנות מונחה עצמים – מטלה 2- ירושה וממשקים

חלק 1 – ממשקים – INTERFACE

א) בשאלה זו נכזר במחלקות Point, Triangle, Rectangle, ShapeContainer
של מטלה אחרונה בקורס מבוא לחישוב .

נגדיר ממשק בשם Drawable:

```
public interface Drawable {
      public boolean equals(Drawable d);
      public boolean contains(Point p);
      public double perimeter();
      public double area();
      public void translate(Point p);
}
```

נגדיר מחדש מחלקות Point, Triangle, Rectangle כך שהן תממשנה את הממשק.
נכתוב מחלקה ShapeContainer כך שהיא תכיל רק מערך אחד של צורות מטיפוס Drawable ובהתאם לזה נשנה את השיטות שלה.
הדרכה: צריך להבחין בין שיטות שבהן צריך הקומפיילר לדעת את הטיפוס האמתי של צורה ושיטות שבהם מספיק להשתמש בטיפוס של Drawable. בדרך כלל ביצירת אובייקט חדש באמצעות בנאי מעתיק צריך להשתמש בטיפוס אמתי של האובייקט.

תיאור של מחלקת ShapeContainer

```
public ShapeContainer()
```

יצירה של אובייקט חדש שאינו מכיל צורות.

```
public ShapeContainer(ShapeContainer other)
```

"העתקה העמוקה" - אוסף הצורות של האובייקט הקיים ישתכפל באובייקט החדש כסדרן.

עליכם לאחסן את אוסף הצורות באמצעות מערך אחד.

למחלקה שני קבועים ציבוריים ומוגדרת בה פעולת גדילה (שיטה פרטית):

```
public static final int INIT_SIZE=10;
public static final int RESIZE=10;
```

הקבוע INIT_SIZE הוא גודל המערכים הראשוני שייווצרו בעת יצירת אובייקט ריק.

הקבוע RESIZE הוא מידת הגדילה של המערכים במידה ואין בהם יותר מקום לקלוט צורות נוספות,

פעולת הגדילה תתבצע רק כאשר יש צורך להוסיף צורה ואין מקום.

```
public int size()  // return the number of stored shapes in the container
public int T_size()        // return the number of stored triangles in the container
public int R_size()        // return the number of stored rectangles in the container

public void add (Drawable d);    // add the given object  to the container
```

<u>שימו לב:</u> פעולת ההוספה **<u>אינה</u>** יוצרת עותק חדש של האיבר המוסף!

```
public void remove(Point p)      // remove triangles & rectangles containing p
public Triangle T_at(int i)      // return a new copy of the triangle number i
public Rectangle R_at(int i)     // return a new copy of the rectangle number i
public double sumArea()          // return the sum of the areas of all the triangles & rectangles


public void translate(Point p) //Translates (mutator) all the shapes by a Point
```

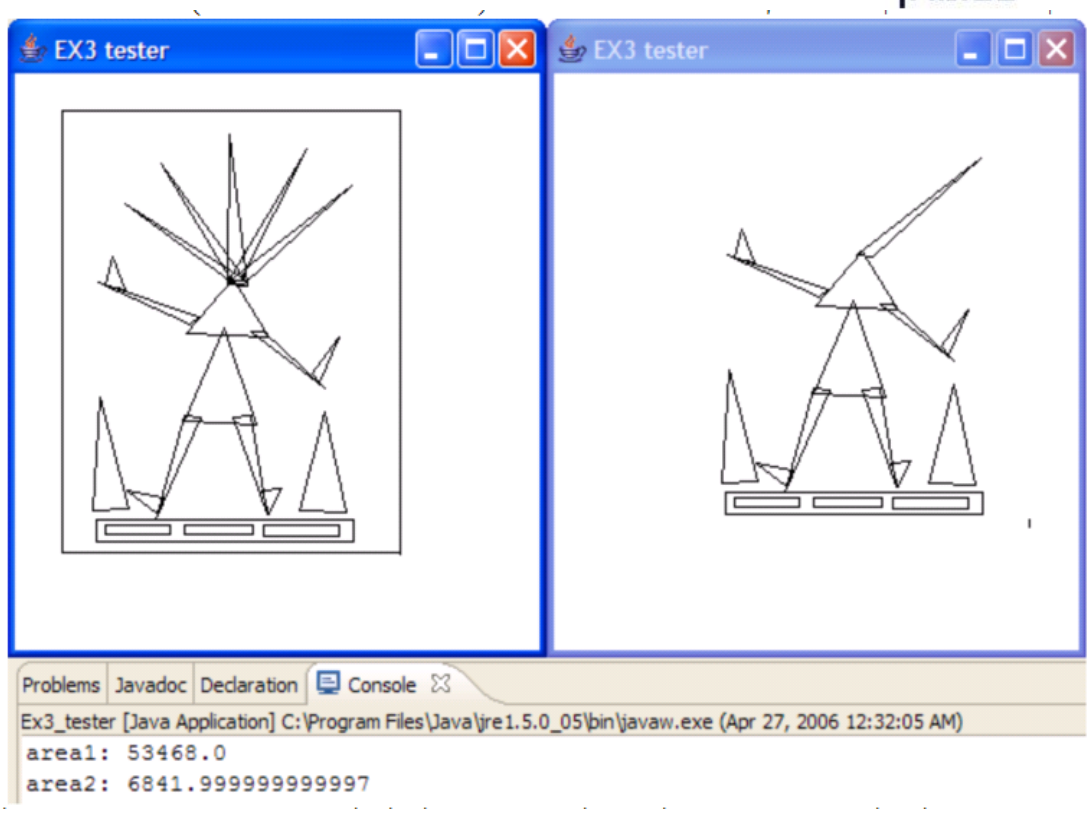
ראו הגדרת translate במחלקה נקודה.

```
public void minMaxPerimeter(int num)// calculates and prints min and max perimeter
                                    // of the shapes (triangles & rectangles)
                                    // and the number of comparisons
```

<u>שימו לב:</u> השתמשו באלגוריתם שמחשב איבר מינימאלי ומקסימאלי **בזוגות** (קורס algorithms1)

<u>הערות:</u>

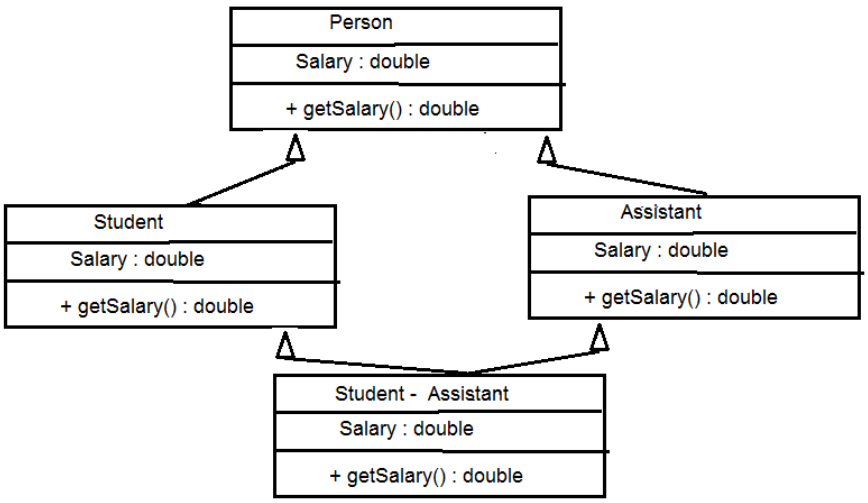
שימו לב למקרי קצה, כלומר הוספה על גבול גודל המערך, ביצוע פעולות עם קלט לא תקין וכיו"ב.
לא נדרש לבצע פעולת "הקטנה" במידה ונמחקות צורות מהאוסף.
לא ניתן להניח כי הפרמטרים המתקבלים תקניים / מאותחלים.
חתימות השיטות חייבות להיות זהות לחלוטין לחתימות המתוארות במסמך זה.
יש לבדוק את התכנית בעזרת קובץ Ex4_tester.java.
בדקו שתוכנית הבדיקה שלכם אכן מציירת את החלונות הבאים (ומחשבת את השטחים כאמור).




```
size1: 23
size2: 18
number of comparision1: 31
max perimeter1: 860.0
in perimeter1: 8.0
number of comparision2: 25
max perimeter2: 310.0
min perimeter2: 8.0
```

ב) ממשקים ב- JAVA 8

בתכנות מונחה עצמים הורשה מרובה היא תכונה של שפות תכנות, כמו למשל C++, בהם מחלקה יכולה לרשת מאפיינים ותכונות ממספר מחלקות אחרות. הורשה מרובה גוררת בעיה הנקראת בעיית היהלום (diamond problem), כאשר מחלקות Student, יורשות מ-Person, ומחלקת Assistant יורשת מ- Person וסטודנט שעובד כמתרגל צריך לרשת משתי המחלקות: Student ו-Assistant :



ב-JAVA אין הורשה מרובה, אבל המחלקה יכולה לממש מספר ממשקים. בגרסה חדשה של JAVA, ב- JAVA 8 הוסיפו לממשקים תכונה חדשה: default method. הקוד הבא מהווה בעיה דומה לביית היהלום. עליכם למצוא לפחות שלושה פתרונות לבעיה זו.

```
2
3  interface Alpha{
4      public default int methodA(){
5          int result = 0;
6          System.out.println("Print from Alpha methodA");
7          return result + 4;
8      }
9  }
10
11 //*********************************************************
12
13 interface Betta extends Alpha{
14     public default int methodA(){
15         int result = 0;
16         System.out.println("Print from Betta methodA");
17         return result + 8;
18     }
19 }
20
21 //*********************************************************
22
23 interface Gamma extends Alpha{
24     public default int methodA(){
25         int result = 0;
26         System.out.println("Print from Gamma methodA");
27         return result + 16;
28     }
29 }
30
31 //*********************************************************
32
33 public class Delta implements Betta,Gamma{
34
35 Duplicate default methods named methodA with the parameters () and () are inherited from the types Gamma and Betta
36
37     public static void main(String args[]){
38         Delta cObj=new Delta();
39         cObj.methodA();
40     }
41 }
```

חלק 2 – Exceptions

תרגיל זה כבר עשיתם בקורס מבוא לחישוב. עכשיו עליכם להוסיף מחלקה SquareEquationException שתטפל בקלט לא תקין. ובכל מקרה שהפתרון אינו קיים צריך לזרוק חריגה עם הודעה מתאימה ולחזור לקליטת מקדמים.

א)

על המשתמש להזין שלושה מספרים ממשיים(a,b,c), שמייצגים את מקדמי משוואה ריבועית:
ax^2 + bx + c = 0. לאחר מכן על התוכנית להדפיס:
אם יש שני פתרונות שונים (ממשיים) את שני ה פתרונות: x1=… , x2=…
אם קיים פתרון יחיד למשווה: x1=x2=…
אם למשוואה אין פתרון צריך לזרוק חריגה עם הודעה מתאימה. תכנית בדיקה:

```
public static void main(String[] args) {
      SquareEquation.sqEquation();
}
```

תוצאות ההרצה:

```
aX^2+bX+c=0: Enter a,b,c:
Enter a: -2.3
Enter b: 5.1
Enter c: -12.62
-2.3X^2+5.1X+-12.62=0:
SquareEquationException: Error: NO real roots!
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
      at SquareEquation.sqEq(SquareEquation.java:26)
      at SquareEquation.sqEquation(SquareEquation.java:42)
      at SquareEquation.main(SquareEquation.java:53)
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: -2.3
Enter b: 5.1
Enter c: 12.98
-2.3X^2+5.1X+12.98=0:
x1:-1.5128848463076623    x2:3.730276150655489
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: 1
Enter b: -5
Enter c: 6
1.0X^2+-5.0X+6.0=0:
x1:3.0    x2:2.0
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
```

```
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: 1
Enter b: -2
Enter c: 1
1.0X^2+-2.0X+1.0=0:
x1=x2:1.0
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: 0
Enter b: 2
Enter c: 5
0.0X^2+2.0X+5.0=0:
x1=:-2.5
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: 0
Enter b: 0
Enter c: 3
0.0X^2+0.0X+3.0=0:
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
SquareEquationException: Error, no answer!!
      at SquareEquation.sqEq(SquareEquation.java:14)
      at SquareEquation.sqEquation(SquareEquation.java:42)
      at SquareEquation.main(SquareEquation.java:53)
1
aX^2+bX+c=0: Enter a,b,c:
Enter a: 0
Enter b: 0
Enter c: 0
0.0X^2+0.0X+0.0=0:
Enter 0 or any number to Exit or 1 to solve aX^2+bX+c=0:
SquareEquationException: x1 can be any number - trivial!
      at SquareEquation.sqEq(SquareEquation.java:15)
      at SquareEquation.sqEquation(SquareEquation.java:42)
      at SquareEquation.main(SquareEquation.java:53)
0
Ex3b - Done!
```

ב)

מעטפת התוכנית, בחלק זה עליכם לכתוב את המסגרת: שתציג למשתמש תפריט לבחירתו:

0 – יציאה מהתוכנית (או כל מספר אחר ששונה מ 1).

1 – סעיף ראשון (חישוב של פתרונות משוואה ריבועית)

לאחר סיום הסעיף (הראשון) על התוכנית לחזור ולהציג את התפריט ההתחלתי.

חלק 3 – Iterators

בחלק זה יש לכתוב מחלקה **גנרית** בשם `LinkedListDouble` המייצגת רשימה מקושרת דו-כיוונית.
למחלקה זו יש לממש ממשק של java.util.ListIterator שמאפשר לעבור על הרשימה שבשני הכוונים ולקבל את המיקום הנוכחי של האיטרטור.

https://docs.oracle.com/javase/7/docs/api/java/util/ListIterator.html

א) המחלקה `LinkedListDouble` צריכה להכיל מתודות:

- **בנאי ללא ארגומנטים,**
- מתודה להוספת איבר חדש
  `public void add(T item){…}`

- מתודה בוליאנית לחיפוש איבר מסוים. המתודה מחזירה `true`, אם האיבר נמצא בתוך הרשימה ומחזירה `false`, אם האיבר אינו נמצא בתוך הרשימה.
  `public boolean contains(T item){…}`

- מתודה למחיקת איבר מסוים. . המתודה מחזירה את האיבר הנמחק, ובמעדה האיבר לא נמצא ברשימה, היא מחזירה `null`.
  `public T remove(T item){…}`

- מתודה שמחזירה אורך הרשימה:
  `public int size(){…}`

- מתודה **`public String toString(){…}`**, המחזירה מחרוזת שמייצגת את איברי הרשימה

ב) יש לממש את הממשק ListIterator כמחלקה פנימית אנונימית של מתודה

```
public ListIterator<T>listIterator(){...}
```

יש לממש את המתודות של ממשק ListIterator , שלא מוגדרות כאופציונליות:

- **hasNext**

`boolean hasNext()`
Returns `true` if this list iterator has more elements when traversing the list in the forward direction. (In other words, returns `true` if `next()` would return an element rather than throwing an exception.)

Specified by: `hasNext` in interface `Iterator<E>`

Returns:
`true` if the list iterator has more elements when traversing the list in the forward direction

- **next**

`E next()`
Returns the next element in the list and advances the cursor position. This method may be called repeatedly to iterate through the list, or intermixed with calls to `previous()` to go back and forth. (Note that alternating calls to `next` and `previous` will return the same element repeatedly.)

Specified by: `next` in interface `Iterator<E>`

Returns:

the next element in the list

Throws:

`NoSuchElementException` - if the iteration has no next element

- **hasPrevious**

`boolean hasPrevious()`
Returns `true` if this list iterator has more elements when traversing the list in the reverse direction. (In other words, returns `true` if `previous()` would return an element rather than throwing an exception.)

Returns:

`true` if the list iterator has more elements when traversing the list in the reverse direction

- **previous**

`E previous()`
Returns the previous element in the list and moves the cursor position backwards. This method may be called repeatedly to iterate through the list backwards, or intermixed with calls to `next()` to go back and forth. (Note that alternating calls to `next` and `previous` will return the same element repeatedly.)

Returns:

the previous element in the list

Throws:

`NoSuchElementException` - if the iteration has no previous element

- **nextIndex**

`int nextIndex()`

Returns the index of the element that would be returned by a subsequent call to `next()`. (Returns list size if the list iterator is at the end of the list.)

Returns:

the index of the element that would be returned by a subsequent call to `next`, or list size if the list iterator is at the end of the list

- **previousIndex**

`int previousIndex()`

Returns the index of the element that would be returned by a subsequent call to `previous()`. (Returns -1 if the list iterator is at the beginning of the list.)

Returns:

the index of the element that would be returned by a subsequent call to `previous`, or -1 if the list iterator is at the beginning of the list

מתודות שמוגדרות כאופציונליות יש לממש לפי הדוגמה:

```
@Override
public void add(T arg0) {
      throw new RuntimeException();
}
```

על כל התרגילים יש לכתוב תכנית לבדיקה. תכנית זו לא תיבדק.

בהצלחה רבה!