

תוכנה 1 – סתיו תשע"ד

תרגיל מספר 7

מנשקים Interfaces

הנחיות כלליות:

קראו בעיון את קובץ נהלי הגשת התרגילים אשר נמצא באתר הקורס.

- הגשת התרגיל תעשה במערכת ה-moodle בלבד (<http://moodle.tau.ac.il/>).
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש ומספר התרגיל (לדוגמא, עבור המשתמש aviv יקרא הקובץ aviv_hw7.zip). קובץ ה-zip יכיל:
 - א. קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז.
 - ב. קבצי ה-java של התוכניות אותם התבקשתם לממש, כולל תיקיות החבילה.

חלק א' (40 נק')

בחלק זה נתרגל כתיבת מחלקות המממשות מנשק נתון. לאחר מכן, נשתמש בהן בעזרת תכונת הפולימורפיזם של תכנות מונחה עצמים ב-Java.

נתון המנשק Shape:

```
public interface Shape {

    // Returns the shape's area
    public float getArea();

    // Returns the shape's perimeter
    public float getPerimeter();

    // Returns a string representation of the shape details.
    // The returned string would include the shape's name, its fields
    // and their corresponding values.
    public String getDetails();

}
```

1. עליכם לכתוב שלוש מחלקות המממשות את המנשק Shape. להלן פירוט חתימת הבנאי של כל אחת מהן (כל בנאי מקבל כפרמטרים את הנתונים הנדרשים להגדרת הצורה אותה הוא יוצר):

```
// x and y are the coordinates of the centre
public Circle (int x, int y, int radius)

// x and y are the coordinates of the bottom left corner
public Rectangle (int x, int y, int width, int height)

// x and y are the coordinates of the bottom left corner
public Square (int x, int y, int width)
```

- על כל מחלקה להכיל שדות פרטיים אליהם ניתן לגשת עם מתודות getter ו-setter ציבוריות.

- כל אחת משלוש המחלקות מממשת את המנשק Shape ועל כן צריכה לממש את המתודות המוגדרות בו.
- המתודות `getArea` ו-`getPerimeter` יחזירו את שטח והיקף הצורה בהתאמה (שימו לב שלצורך החישוב יש להשתמש בקבוע `Math.PI` לקבלת הערך פאי).
- המתודה `getDetails` תחזיר מחרוזת המייצגת את פרטי הצורה ותכיל את שם הצורה, ואת כל השדות שבה לצד הערכים שלהם. דוגמא למחרוזות שתוחזרנה עבור מעגל, מלבן וריבוע בהתאמה:

"Circle: X=100, Y=150, Radius=30"

"Rectangle: X=100, Y=150, Width=50, Height=50"

"Square: X=100, Y=150, Width=50"

2. כתבו מחלקה בשם `ShapeDimensionCalculator` המקבלת כארגומנט יחיד בשורת הפקודה מסלול קובץ שימשם לפלט. התוכנית תקבל מהמשתמש (דרך ה-`System.in`) סדרה של נתוני צורות, תחשב עבור כל צורה את שטחה ואת היקפה, ולבסוף תדפיס דו"ח מסכם לקובץ הפלט.

עבור כל צורה שנבחרה ע"י המשתמש, תבקש התוכנית את הנתונים הנדרשים לבנייתה.

דוגמא להרצת התוכנית:

עבור שורת הפקודה הבאה: (קלט משתמש בכחול)

```
java ShapeDimensionCalculator output.txt
```

```
Shape Dimension Calculator

Please choose shape type:
C - Circle
S - Square
R - Rectangle
E - End
C
Please enter X coordinate:
100
Please enter Y coordinate:
150
Please enter radius:
30
Shape added: [Circle: X=100, Y=150, Radius=30]

Please choose shape type:
C - Circle
S - Square
R - Rectangle
E - End
S
Please enter X coordinate:
```

```

200
Please enter Y coordinate:
230
Please enter width:
50
Shape added: [Square: X=200, Y=230, Width=50]

Please choose shape type:
C - Circle
S - Square
R - Rectangle
E - End
E

Report saved to file output.txt.

```

לאחר ריצת התוכנית כמודגם לעיל יכיל קובץ הפלט output.txt את הטקסט הבא:

```

Shape Dimension Calculator

Circle: X=100, Y=150, Radius=30   Area: 2827.4333882308138   Perimeter: 188.49555921538757
Square: X=200, Y=230, Width=50   Area: 2500.0   Perimeter: 200.0

Total Number of Shapes: 2

Total Area Sum: 5327.433388230814

Total Perimeter Sum: 388.4955592153876

```

הערות:

- המשתמש יכול להכניס 20 צורות לכל היותר. לאחר הכנסת הצורה ה-20 התוכנית תצא אוטומטית משלב הכנסת הנתונים ותייצר את הדו"ח המסכם לקובץ הפלט.
- ניתן להניח שהמשתמש מכניס קלט חוקי (אותיות או מספרים שלמים כנדרש).
- יש לאחסן את הצורות במערך מסוג Shape ולבצע את החישובים בגמר קליטת הנתונים.
- בקובץ הפלט יש 3 עמודות עבור כל צורה המופרדות ע"י טאבים. העמודה הראשונה היא תוצר של מתודת ה-getDetails של הצורה, השניה מציגה את השטח והשלישית את ההיקף.

3. להלן מנשק נוסף בשם Scalable:

```

public interface Scalable {

    // Scales the shape by the given x and y offsets
    public void scale (float scaleFactor);

}

```

המתודה scale מקבלת שבר שימשם לשינוי יחסי של מימדי הצורה.

- שנו את המחלקות Circle, Rectangle ו-Square כך שיממשו גם את המנשק Scalable ע"י מימוש המתודה scale בכל אחת מהמחלקות.
- המחלקה Circle תממש מתודה זו ע"י הכפלת רדיוס המעגל בפקטור הנתון. המחלקה Rectagle תממש מתודה זו ע"י הכפלת ה-width וה-height בפקטור הנתון. המחלקה Square תכפיל את ה-weight בפקטור הנתון.
- היעזרו ב-Math.round ליעגול הערך המתקבל לפני המרה ל-int.
- כתבו מתודה בשם scaleShapes המקבלת מערך צורות המקיימות את המנשק הנ"ל, ופקטור שינוי מימדים. המתודה תשנה את מימדי כל הצורות במערך לפי הפקטור הנתון. יש לממש את המתודה בתוך המחלקה ShapeDimensionCalculator. חתימת המתודה תהיה:

```
public static void scaleShapes(Scalable[] scalableShapes, float scaleFactor);
```

שימו לב: בחלק זה יש להגיש את הקבצים הבאים (ניתן להוסיף קבצי עזר בהם השתמשתם):

ShapeDimensionCalculator.java, Circle.java, Rectangle.java, Square.java

חלק ב' (60 נק')

המנשק IPAddress המופיע למטה מייצג כתובת של Internet Protocol (IP). דוגמאות לכתובות IP הן:

127.0.0.1
192.168.1.10

כתובת IP, כפי שנתן לראות, מורכבת מארבעה חלקים. ערכו של כל אחד מהחלקים הוא מספר שלם בין 0 ל-255. נממש את המנשק בעזרת שלושה ייצוגים שונים: הראשון עושה שימוש במחרוזות, השני במערך של מספרים מסוג short ואילו השלישי משתמש ב-int יחיד (הסבר מפורט בהמשך).

- א. כתבו **שלוש** מחלקות שונות המממשות את המנשק IPAddress המוגדר למטה:
1. מחלקה בשם IPAddressString, המממשת את המנשק בעזרת ייצוג פנימי של String.
 2. מחלקה בשם IPAddressShort, המממשת את המנשק בעזרת ייצוג פנימי של מערך בגודל 4 של short. כל תא במערך יחזיק מספר בתחום 0..255.
 3. מחלקה בשם IPAddressInt, המממשת את המנשק בעזרת int יחיד.

לכל אחת מהמחלקות יהיה בנאי המתאים לייצוג הפנימי שלה וכמובן כל אחת מהן מממשת את המנשק. ניתן להניח בחוזה שהבנאים מקבלים קלט תקין ליצירת כתובת IP (בהתאם לייצוג הפנימי של כל מחלקה).

```
public interface IPAddress {

    /**
     * Returns a string representation of the IP address, e.g.
     * "192.168.0.1"
     */
    public String toString();

    /**
     * Compares this IPAddress to the specified object
     * @param other
     *         the IPAddress to compare the current against
     * @return true if both IPAddress objects represent the same
     *         IP address, false otherwise.
     */
    public boolean equals(IPAddress other);

    /**
     * Returns one of the four parts of the IP address. The parts
     * are indexed from left to right. For example, in the IP
     * address 192.168.0.1 part 0 is 192, part 1 is 168,
     * part 2 is 0 and part 3 is 1.
     * (Each part is called an octet as its representation
     * requires 8 bits.)
     * @param index
     *         The index of the IP address part (0, 1, 2 or 3)
     * @return The value of the specified part.
     */
    public int getOctet(int index);

}
```

להלן פירוט לגבי אופן מימוש הייצוגים השונים :

1. מחרוזת – יש להשתמש במחרוזת יחידה לצורך ייצוג כתובת ה IP . כל הפעולות יבוצעו בעזרת מחרוזת זו.
2. מערך – כל אחד מחלקי הכתובת (מספר שלם 0-255) יוחזק בתא במערך.
3. int – נשים לב שכל אחד מחלקי כתובת ה-IP הוא מספר שלם בתחום 0-255 (כולל), לפיכך ניתן לייצג אותו בעזרת 8 ביטים. על כן, את ארבעת חלקי הכתובת ניתן לייצג באמצעות 32 ביטים (4 בתים) וזהו בדיוק גודלו של int.

לפיכך, נשתמש ב-int לא כמספר, אלא כרצף בינארי של 32 ביטים. לדוגמא, הכתובת 127.0.0.1 תיוצג ע"י רצף הביטים 01111111000000000000000000000001.

החלק הראשון ייוצג ע"י הביטים במקומות 0-7 (משמאל לימין), החלק השני ע"י הביטים 8-15, השלישי ע"י 16-23 והרביעי ע"י ביטים 24-31.

```

127 <- 01111111 (ביטים 0-7 משמאל לימין)
0 <- 00000000 (ביטים 8-15)
0 <- 00000000 (ביטים 16-23)
1 <- 00000001 (ביטים 24-31)

```

הנחיה: על מנת להשתמש בייצוג זה, עליכם לדעת איך לחלץ את ערכו של כל בית (8 ביט) מהרצף הבינארי באורך 4 הבתים (32 ביטים) שמרכיב את ה-int. נציע 2 דרכים אפשריות:

1. שימוש באופרטורים על ביטים (<<, >>, ~, &, |) להזזה או למיסוך של הביטים ברצף הבינארי כך שיאופסו כל הביטים מלבד אלו השייכים לבית הרצוי.

2. שימוש במחלקה [ByteBuffer](#)

- צרו אובייקט בגודל 4 בתים ע"י `ByteBuffer.allocate(4)`
- היעזרו במתודות `put(i)/get(i)` להצבה ולחילוץ של בית במיקום i.
- שימו לב שהמתודה `get(i)` תחזיר את הבית באינדקס i באובייקט ה-`ByteBuffer` שיצרתם כמשתנה מטיפוס `byte` עם טווח ערכים של `-128..+127`.

כדי לבצע המרה של בית b מטיפוס `byte` למשתנה מטיפוס `int` עם טווח הערכים 0..255. לו אנו זקוקים, ניתן להשתמש בטריק הבא:

```
int i = (int) (b & 0xFF);
```

הערה חשובה: עליכם לממש את המתודות באופן שונה בכל מחלקה בהתאם לייצוג הפנימי. אין להמיר את הייצוג הפנימי לייצוג אחר לצורך מימוש פעולה (רק לצורך פלט). המחלקות השונות לא "ייעזרו" זו בזו (למשל, אסור להשתמש ב-`IPAddressString` על מנת לממש את `IPAddressInt`).

בנוסף, ממשו את המחלקה IPAddressFactory המגדירה את המתודות הבאות:

```
public class IPAddressFactory {
    public static IPAddress createAddress(String ip) {
        ...
    }
    public static IPAddress createAddress(short[] ip) {
        ...
    }
    public static IPAddress createAddress(int ip) {
        ...
    }
}
```

כל אחת מהמתודות הסטטיות יוצרת אובייקט מטיפוס IPAddress, כשהאובייקט הקונקרטי נקבע על סמך טיפוס הקלט.

הערה: מחלקה שתפקידה היחיד הוא יצור אובייקטים של מחלקות אחרות נקראת *factory class*. מחלקות אלו מסתירות את פרטי יצור האובייקטים מלקוחות של אובייקטים אלו. השימוש בטכניקה זו נועד להסתיר את המחלקות הקונקרטיות שמממשות מנשק.

להלן תכנית המדגימה את השימוש במחלקה IPAddressFactory ובמנשק.

```
public class TestIPAddress {
    public static void main(String[] args) {
        int address1 = -1062731775; // 192.168.0.1
        short[] address2 = { 10, 1, 255, 1 }; // 10.1.255.1

        IPAddress ip1 = IPAddressFactory.createAddress(address1);
        IPAddress ip2 = IPAddressFactory.createAddress(address2);
        IPAddress ip3 = IPAddressFactory.createAddress("127.0.0.1");

        for (int i = 0; i < 4; i++) {
            System.out.println(ip1.getOctet(i));
        }

        System.out.println("equals: " + ip1.equals(ip2));
    }
}
```
