

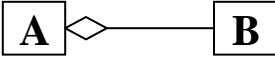
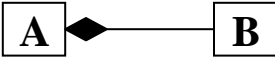
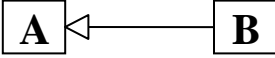
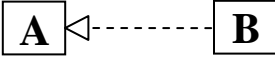
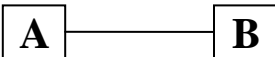
תוכנה 1

תרגיל מספר 9

הנחיות כלליות:

- קראו בעיון את קובץ נוהלי הגשת התרגילים אשר נמצא באתר הקורס.
- הגשת התרגיל תעשה במערכת ה VirtualTAU בלבד (<http://virtual2002.tau.ac.il/>).
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש (לדוגמא, עבור המשתמש zvainer יקרא הקובץ zvainer.zip) קובץ ה zip יכיל:
 - א. קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז. הזהות שלכם.
 - ב. קבצי ה java של התוכניות אותם התבקשתם לממש.
 - ג. קובץ טקסט עם העתק של כל קבצי ה java
 - ד. קובץ טקסט בשם answers עם התשובות לשאלות

הערה כללית: בתרגיל זה אתם מתבקשים, בין היתר, לשרטט דיאגרמות של מחלקות. השתמשו בסיומונים הבאים בלבד:

-  (aggregation) יחס של הכלה (למשל, ל-A יש שדה מטיפוס B)
-  (aggregation) יחס של הרכבה (כאשר B תלוי ב-A. למשל, מחלקה B היא מחלקה פנימית של A)
-  ירושה (B מחלקה היורשת את A):
-  מימוש (B מחלקה המממשת ומנשק היורש את המנשק A):
-  (association) קשר כללי שאינו נופל בקטגוריות הקודמות. למשל, A משתמש במשתנה מטיפוס B במתודה פנימית.

יש לציין: בתוך כל מלבן <<interface>>, <<abstract>> או <<class>>, ואת שם המנשק או המחלקה.

אין צורך לציין: מספרים ושמות שדות על יחסי אגרגציה ואסוציאציה; שמות מתודות ושדות בתוך מלבני המחלקות; יחסים "עקיפים" בין מחלקות (כלומר, אם C יורש מ-B שיורש מ-A, אין צורך לציין קשר בין C ל-A אלא אם יש ביניהם קשר ישיר בנוסף, למשל של הכלה)

יצירת הדיאגרמות: ניתן לעשות זאת דרך Word, PowerPoint, תוכנת הציור המועדפת עליכם או לסרוק שרטוט (בכתב ברור!). כדי למנוע בעיות formatting שונות הפורמט המועדף עבור [answers.pdf](#) הוא [answers.pdf](#).

חלק א':

ברצוננו לתאר בתוכנה מגוון של ביטויים חשבוניים. ביטוי חשבוני הוא ישות הניתנת לשערוך כגון מספר או פעולה חשבונית המופעלת על שניים או שלושה ביטויים חשבוניים (הגדרה רקורסיבית). כל ביטוי חשבוני יודע להדפיס את עצמו.

עליכם להגדיר את הטיפוסים הבאים ולממשם כמחלקות קונקרטיות, מחלקות מופשטות או מנשקים. בנוסף הציגו תרשים מחלקות המתאר את היחס בין הטיפוסים שהגדרתם (מכונה גם עץ הורשה או היררכית מחלקות) בקובץ answers.

- **Expression** – טיפוס המייצג ביטוי כלשהו.
- **Literal** – טיפוס המתאר מספר ממשי (double).
- **BinaryExpression** – טיפוס המתאר פעולה בינארית (פעולה על שני ביטויים).
- **TrenaryExpression** – טיפוס המתאר פעולה טרינרית (פעולה על שלושה ביטויים).
- **Sum** – טיפוס המתאר את פעולת סכום של שני ביטויים.
- **Product** – טיפוס המתאר את פעולת המכפלה של שני ביטויים.
- **Exponent** – טיפוס המתאר את פעולת החזקה של שני ביטויים.
- **Conditional** – טיפוס המתאר פעולה על שלושה פרמטרים x, y, z כך שאם ערכו של x שונה מ-0 יוחזר ערכו של y , ואחרת יוחזר ערכו של z .

בידקו את עצמכם ע"י קוד הלקוח הבא:

```
public class ExpClient {  
  
    public static void main(String[] args) {  
        Expression l1 = new Literal(1.0);  
        Expression l2 = new Literal(2.0);  
        Expression l3 = new Literal(3.0);  
  
        Expression sum = new Sum(l1, l2);  
        Expression e1 = new Exponent(l3, sum);  
        Expression prod = new Product(l1, l2);  
        Expression exp = new Exponent(l2, l3);  
        Expression e2= new Conditional(sum, prod, exp);  
  
        System.out.println(e1 + " = " + e1.eval());  
        System.out.println(e2 + " = " + e2.eval());  
    }  
}
```

פלט התוכנית הוא :

$$(3.0 \wedge (1.0 + 2.0)) = 27.0$$

$$((1.0 + 2.0) ? (1.0 * 2.0) : (2.0 \wedge 3.0)) = 2.0$$

הערות:

- כל אחד משמנות הטיפוסים לעיל יגדיר את המתודה: `public double eval();`
- כל הפעולות שהוגדרו פועלות על ביטויים (Expression). שערך של ביטוי מתבצע ע"י הפונקציה `eval`.
- ביטויים מורכבים ישוערכו ע"י הפעלה רקורסיבית של `eval` על הארגומנטים.
- תשובתכם צריכה לבטא **שימוש חוזר** ברכיבי תוכנה **ולמזער את שכפול הקוד**. הדבר יעשה, בין השאר, ע"י בחירה נכונה של מחלקות קונקרטיות, מחלקות מופשטות ומנשקים. קוד עובד הוא תנאי הכרחי אך לא מספיק במקרה זה.
- שימו לב למתודה `toString()` ולהדפסות הסוגריים. תזכורת: כאשר אופרטור ה- '+' ופונקציית ההדפסה מוצאים עצם שאינו `String` במקום שבו אמור היה להימצא `String`, מופעלת המתודה `toString()` של אותו העצם. במחלקה `Object` קיים מימוש ברירת מחדל של מתודה זו.

חלק ב':

ברצוננו להוסיף למחלקות קיימות את האפשרות להשוות בין שני עצמים מאותה המחלקה. על ההשוואה לתמוך בששת 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, MyComparablePoint, המייצגת נקודות במישור שניתן להשוות ביניהן. קריטריון ההשוואה הוגדר להיות שיעור ה-x של הנקודות, כלומר נקודה היא קטנה מנקודה אחרת עם ערך ה-x שלה קטן מערך ה-x של הנקודה האחרת. במקרה שלשתי הנקודות שיעור ה-x זהה ההשוואה תעשה לפי שיעור ה-y. להלן הקוד המלא של המחלקה הקונקרטית 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;
        return (x != otherPoint.x) ? x < otherPoint.x
            : y <= otherPoint.y;
    }

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

    protected int x, y;
}
```

1. כדי למנוע את שכפול הקוד הגדירה אותה מתכנתת מחלקה מופשטת (abstract class), **AbstComparable**, המממשת את הממשק **MyComparable** ומפשטת את הגדרת המחלקות הקונקרטיות. ממשו את המחלקה **AbstComparable** המופיעה בקוד למטה. על המחלקה להיות כללית ככל האפשר (ולא מותאמת ספציפית למחלקה **MyComparablePoint**). כך, שעיי כתיבת מחלקות יורשות המממשות את **lessThanEqual** (ומבלי לדרוס מתודות נוספות), נוכל, למשל, להשוות בין מלבנים לפי שטחם, בין אנשים לפי הסדר האלפאביתי של שמם, בין מאורעות לפי תאריך התרחשותם ועוד ועוד. הימנעו ככל הניתן משכפול קוד:

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

2. 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 נקודה (x, y) מוגדרת להיות $\sqrt{x^2 + y^2}$. ממשו את המחלקה **MyNormComparablePoint** בעזרת ירושה. שימו לב – על המחלקה לתמוך בכל השירותים שאותם סיפקה **MyComparablePoint**.

3. למימוש המחלקה **MyNormComparablePoint** בעזרת ירושה חסרונות ויתרונות. מהי הבעייתיות המרכזית של מימוש זה ומהו היתרון המרכזי שלו? **השלימו את התשובה בקובץ answers**

4. הציעו מימוש חלופי למחלקות **MyComparablePoint** ו- **MyNormComparablePoint** שיפתור את הבעייתיות שיוצרת הירושה במקרה זה (שציינתם בסעיף הקודם). 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. **השלימו את התשובה בקובץ answers**

הערה:

את הממשק **MyComparable**, המחלקה **MyComparablePoint** ושלדים עבור **AbstComparable** ו- **MyNormComparablePoint** תוכלו למצוא באתר הקורס.

בהצלחה !