

תרגיל 10 בקורס תוכנה 1

הוראות הגשה:

1. קראו בעיון את [קובץ נוהלי הגשת התרגילים](#) אשר נמצא באתר הקורס.
2. הגשת התרגיל תעשה ע"י המערכת VirtualTAU
3. הגשת התרגיל תבצע ע"י יצירת קובץ zip שנושא 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 התכניות שהתבקשתם לכתוב.
 - ג. קובץ טקסט עם העתק של כל קבצי ה Java.
 - ד. קובץ טקסט עם התשובות לסעיפים ג' ו-ד' בחלק 3.

חלק 1: הורשה

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 (כן – ההגדרה רקורסיבית). כל ביטוי חשבוני יודע להדפיס את עצמו.

עליך להגדיר את הישויות הבאות ולממשן, כמחלקות קונקרטיות, מחלקות מופשטות או מנשקים. בנוסף הציגו תרשים מחלקות המתאר את היחס בין הישויות שהגדרתם (מכונה גם עץ הורשה או היררכית מחלקות).

- **Expression** – ישות המייצגת ביטוי כלשהו.
- **Literal** – ישות המתארת מספר ממשי בודד (double).
- **BinaryOp** – ישות המתארת פעולה בינארית (פעולה על שני ביטויים).
- **TrenaryOp** – ישות המתארת פעולה טרינרית (פעולה על שלושה ביטויים).
- **Sum** – ישות המתארת את פעולת הסכום.
- **Product** - ישות המתארת את פעולת המכפלה.
- **Exponent** - ישות המתארת את פעולת החזקה.
- **CondExp** – ישות המתארת פעולה על שלושה פרמטרים a, b, c שהיא $a ? b : c$. שערוך הביטוי מתבצע כך: אם ערכו של a שונה מ 0.0 יוחזר ערכו של b אחרת יוחזר ערכו של c.

בדקו 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 CondExp(sum, prod, exp);  
  
        System.out.println(e1 + " = " + e1.eval());  
        System.out.println(e2 + " = " + e2.eval());  
    }  
}
```

פלט התוכנית הוא:

```
((3.0) ^ ((1.0) + (2.0))) = 27.0  
((1.0) + (2.0)) ? ((1.0) * (2.0)) : ((2.0) ^ (3.0)) = 2.0
```

הערות:

- כל אחת מ-8 הישויות לעיל תכיל את המתודה: `public double eval();`
- כל הפעולות לעיל פועלות על ביטויים (Expression), שערך של ביטוי מתבצע ע"י הפונקציה `eval`.
- ביטויים מורכבים ישוערכו ע"י הפעלה רקורסיבית של `eval` על הארגומנטים.
- תשובתך 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 2: Sorted Set Iterator

ניתן לכם מנשק גנרי SortedSet המייצג אוסף של עצמים ממוינים ושונים אחד מן השני (זהו מנשק שונה ממנשק בשם זהה הקיים ב-JDK). מאחר והעצמים נשמרים בצורה ממוינת במימושים של מנשק זה, איטרטור על האוסף יחזיר אותם אחד-אחד בסדר עולה. בנוסף ניתן לכם מימוש פשוט של המנשק, מחלקה בשם SimpleSortedSet. מחלקה זו תשמש אתכם אך ורק כדוגמה ושאר בדיקות שתמצו לעשות על המימושים שלכם.

חשוב לשים לב כי כל הקבוצות שנגדיר הינן בלתי ניתנות לשינוי, כלומר העצמים בתוכן נקבעים בעת היצירה. לא ניתן להוסיף/להוריד עצמים לאחר מכן מן הקבוצות. תכונה זו תאפשר לכם לשמור תוצאות של חישובים ארוכים יחסית (ערך מקסימום, ערך מינימום וגודל) ולא לחשב אותם בכל פעם מחדש.

שימו לב לאיטרטור SortedIterator, הרחבה ריקה של Iterator שמסמלת כי האיטרטור מחזיר ערכים ממוינים.

תיעוד עבור המחלקות וכן הקוד שלהן זמין באתר.

עליכם לממש את המחלקות הבאות בשני אופנים שונים כפי שיוסבר בהמשך. המחלקות אותן עליכם לממש הן:

א. **BinaryOpSortedSet** – עליכם לתכנן ולממש מחלקה מופשטת. מחלקה מופשטת זו מייצגת קבוצה ממוינת שהיא תוצר של פעולה, כגון איחוד או חיתוך על קבוצות ממוינות. יש לממש במחלקה זו את כל הקוד המשותף שניתן. השירותים אותם אתם חייבים לממש באופן מלא במחלקה זו הינם:

- `int size();`
- `boolean isEmpty();`
- `T getMinimum();`
- `T getMaximum();`

בהתאם לדרישות תכנון נוספות שיוסברו בהמשך תצטרכו לשקול ולהחליט אם ואילו שירותים נוספים ימומשו במחלקה זו. ניתן להוריד "שלד" של המחלקה מהאתר.

ב. **IntersectionSortedSet** – מחלקה זו מייצגת את תוצר פעולת החיתוך בין שתי קבוצות. המחלקה נבנית באמצעות שתי קבוצות ממוינות A ו-B, ומאפשרת גישה רק לעצמים בחיתוך של A ו-B. במחלקה זו תממשו בנאי וכן את הפונקציות המוגדרות במנשק שטרם מומשו במחלקה האבסטרקטית.

ג. **UnionSortedSet** – מחלקה זו מייצגת את תוצר פעולת האיחוד בין שתי קבוצות. המחלקה נבנית באמצעות שתי קבוצות ממוינות A ו-B, ומאפשרת גישה רק לעצמים באיחוד של A ו-B. במחלקה זו תממשו בנאי וכן את הפונקציות המוגדרות במנשק שטרם מומשו במחלקה האבסטרקטית.

ד. **SymetricDifferenceSortedSet** – מחלקה זו מייצגת את תוצר פעולת ההפרש הסימטרי בין שתי קבוצות. המחלקה נבנית באמצעות שתי קבוצות ממוינות A ו-B, ומאפשרת גישה רק לעצמים שנמצאים באיחוד של A ו-B ואינם נמצאים בחיתוך שלהן (ההפרש בין קבוצת האיחוד לקבוצת החיתוך) מחלקה זו תממשו בנאי וכן את הפונקציות המוגדרות במנשק שטרם מומשו במחלקה האבסטרקטית.

נתאר שתי גישות מימוש אפשריות. עליכם לממש את היררכית המחלקות המתוארת לעיל פעמיים, עבור כל אחת מהגישות השונות.

גישה א' – eager:

בגישה זו, בעת יצירת אובייקט של אחת המחלקות המתארות איחוד, חיתוך או הפרש סימטרי בין קבוצות נחשב את תוצאת הפעולה באופן מיידי ונשמור אותה. מכאן ואילך כל הפעולות המוגדרות במנשק יוכלו לפעול על ייצוג התוצאה (כפי שתבחרו אותו).

גישה ב' – lazy:

בגישה זו לא נחשב כלום בעת יצירת האובייקט, אלא נשמור את הקבוצות שעליהן אנחנו פועלים (מובטח לנו שהן לא ישתנו בהמשך). כל הפעולות ימומשו בעזרת האיטרטורים של שתי קבוצות הקלט בלי שבשלב מסוים נחשב את קבוצת התוצאה בכללותה.

את התוצאות של `size`, `minimum` ו-`maximum` נחשב רק כאשר נדרש להן בפעם הראשונה. את התוצאות של שלושת הפונקציות האלה בלבד מותר לכם לשמור כדי שבפעמים הבאות שהן יקראו תוכל להחזיר את הערך השמור ולא לבצע את כל החישוב שעלול להיות יקר (memoization).
בכל אחת מהמחלקות נממש איטרטור חכם שכאשר נקדם אותו הוא ייתן את האשליה שהוא מתקדם על קבוצת התוצאה ולא על קבוצות הקלט. את האיטרטורים יש לממש כמחלקות פנימיות של מחלקות הקבוצה השונות. שימו לב שאם יש שכפול קוד בין האיטרטורים עליכם להעבירו למחלקה משותפת שבאופן טבעי תהיה מחלקה פנימית במחלקה האבסטרקטית `BinaryOpSortedSet`. יתכן כי מחלקת האיטרטור המשותפת תהיה אף היא אבסטרקטית.

הגשה:

לצורכי הגשה שנו את שם המחלקה (והקובץ) כך שיהיה ברור אם מחלקה זו שייכת למימוש `lazy` או למימוש `eager`. הוסיפו את התחילית `Lazy` או `Eager` לשם המחלקה.
לדוגמה עבור המחלקה `BinaryOpSortedSet` תממשו את המחלקות `LazyBinaryOpSortedSet` ו-`EagerBinaryOpSortedSet`. עבור המחלקה `UnionSortedSet` ממשו את המחלקות `LazyUnionSortedSet` ו-`EagerUnionSortedSet` וכך גם עבור שאר המחלקות.

חלק 3 – ComparablePoint

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 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;  
}
```

א. 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 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 lessThan (MyComparable other) {...}
    public boolean greaterThanEqual (MyComparable other) {...}
    public boolean greaterThan (MyComparable other) {...}
    public boolean equal (MyComparable other) {...}
    public boolean notEqual (MyComparable other) {...}

}
```

ב. 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**.

ג. למימוש המחלקה **MyNormComparablePoint** בעזרת ירושה חסרונות ויתרונות. מהו החסרון המרכזי של מימוש זה ומהו היתרון המרכזי שלו? **השלימו את התשובה בקובץ txt (או doc) וצרפו להגשה**

ד. 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 תרשימים מחלקות (מלבנים וחיצים) לעיצוב החדש. אין צורך לציין בתרשימים פרטים שאינם מהותיים לעיצוב החדש. **השלימו את התשובה בקובץ txt (או doc) וצרפו להגשה**