

01.01.2006

תוכנה 1 בשפת Java
בית הספר למדעי המחשב
אוניברסיטת תל אביב

תרגיל בית מספר 5

תאריך הגשה: 15.1.2006

שאלה 1

עבור טיפוס T שהוא Integer, נרצה לשכלל את FixedCapacityQueue<T> מתרגיל 3, כך שיתמוך בשרותי סכום ומוצע של אברי התור הנוכחיים. המחלקה החדשה לא תהיה כללית (Generic) ותשמש לאחסון אברים מטיפוס Integer בלבד.

המחלקה החדשה תתמוך בכל שרותי המחלקה המקורית כפי שהוגדרו בתרגיל 3 ובנוסף תתמוך בשני השרותים הבאים:

```
/** returns the current sum of all the elements in the queue. The sum
 * of the empty queue is zero
 */
public int sum();

/** returns the current average of all the elements in the queue.
 * This operation can be applied only if the queue is not empty
 */
public double average();
```

יש לממש את המחלקה החדשה בשתי דרכים שונות:

1) האצלה (delegation) של FixedCapacityQueue<Integer>:

יש ליצור מחלקה בשם DelegatingFixedCapacityQueue שתכיל בתוכה שדה מטיפוס FixedCapacityQueue<Integer>. שדה זה ישמש כשפק של המחלקה החדשה. כאשר ידרשו מהמחלקה DelegatingFixedCapacityQueue שרותים שסיפקה המחלקה המקורית היא 'תפנה' אותם למחלקה המקורית.

2) ירושה (extension) מ-FixedCapacityQueue<Integer>:

יש ליצור מחלקה בשם ExtendedFixedCapacityQueue שתירש מהמחלקה FixedCapacityQueue<Integer>.

שימו לב, לא ניתן לערוך שינויים בדרישות המחלקה FixedCapacityQueue<T> המקורית. המחלקות DelegatingFixedCapacityQueue ו-ExtendedFixedCapacityQueue צריכות לעבוד לא רק עם המחלקה FixedCapacityQueue<T> שתספקו, אלא גם עם כל FixedCapacityQueue<T> שעונה על הדרישות המקוריות. מצב זה שכיח מאוד בחיים האמיתיים, שם אין לכם שליטה על המחלקות שמהם אתם יורשים, ולא ניתן לתקן אותן בדיעבד.

שאלה 2

בשאלה זו עליך לתת Design (Prototypes) בלבד.

מנהל המרינה בעיר הנמל הציורית גוציק ביקש ממך לממש אפליקציה לניהול כלי השייט העוגנים במרינה.

כלי השייט נחלקים ל:

- 1) סירות מרוץ
- 2) סירות משוטים
- 3) יאכטות

עבור כל כלי שייט יש לשמור את הפרטים הבאים:

- א) שם כלי השייט
- ב) אורך
- ג) רוחב
- ד) משקל
- ה) שנת יצור
- ו) שם החברה שיצרה את כלי השייט
- ז) החומר ממנו בנוי כלי השייט – יכול להיות: עץ, ברזל, פיברגלס או בטון

עבור סירות מרוץ יש לשמור גם מהירות מקסימלית. עבור סירות משוטים יש לשמור גם את מספר המשוטים. עבור יאכטות יש לשמור גם את מחירן. יאכטות נחלקות לשני סוגים: יאכטות מנועיות ויאכטות מפרש. עבור יאכטות מנועיות יש לשמור גם את עוצמת המנוע (בכוחות סוס), ועבור יאכטות מפרש יש לשמור את מספר התרנים, גובה תורן מקסימלי ושטח המפרשים הכולל. כמו כן, קיימות במרינה מספר יאכטות משולבות (דהיינו יאכטות מנועיות עם מפרשים). לכל כלי השייט, ממשק המאפשר לעדכן את פרטי כלי השייט.

כל כלי שייט צריך לעגון ברציף. במרינה קיימים 100 רציפי עגינה קטנים (ממוספרים מ-1 עד 100), ו-26 רציפים גדולים (מסומנים ב-A עד Z). בכל רציף יכול לעגון לכל היותר כלי שייט אחד. ברציף קטן יכול לעגון כלי שייט שאורכו עד 20 מטר ובגודל עד 100 מטר. בחלק מהרציפים (בגלל רוח חזקה) אסור לעגון יאכטות עם מפרשים (יש לאפשר למנהל המרינה לעדכן נתון זה עבור כל רציף).

מערכת הניהול מאפשרת:

- 1) להוסיף ולמחוק כלי שייט מהרשימה. כאשר מוסיפים כלי שייט יש להקצות לו רציף מתאים לעגינה (או להודיע אם אין מקום מתאים).
- 2) לעדכן ולהדפיס פרטים של כלי שייט קיים.
- 3) להחליף את מקום העגינה בין שתי יאכטות (או להודיע אם אי אפשר).
- 4) להדפיס את רשימת כל כלי השייט, ממוינים לפי אורכם, ועבור כל כלי שייט לציין גם את רציף העגינה.
- 5) להדפיס את רשימת כל רציפי העגינה המלאים, ממוינים לפי מספרם (או סימנם – עבור רציפים גדולים), ועבור כל רציף לציין את שם כלי השייט שעוגן בו.
- 6) להדפיס את רשימת כל רציפי העגינה הפנויים (שוב, בצורה ממוינת).

עליכם לספק מסמכי Javadoc (מסמכים אלו נקראים גם API, ראשי תבות של: Application Programming Interface) שיתארו את המחלקות הממשיות, המחלקות המופשטות והממשקים הדרושים למימוש התוכנית. כמו כן יש לצרף תרשים מחלקות המתאר את היחסים בין הישויות השונות (ראה <http://www.classdraw.com/Help.htm>) ודיון קצר לגבי הנימוקים ליחסים כפי שנבחרו.

אין צורך לממש בקוד מפורש אף מתודה.

שאלה 3

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 (מחלקות קונקרטיות, מחלקות מופשטות ומנשקים):

1. Expression – ישות המייצגת ביטוי כלשהו.
2. Literal – ישות המתארת מספר בודד (double).
3. BinaryOp – ישות המתארת פעולה בינארית (פעולה על שני ביטויים).
4. TernaryOp – ישות המתארת פעולה טרינארית (פעולה על שלושה ביטויים).
5. Sum – ישות המתארת סכום.
6. Product – ישות המתארת מכפלה.
7. Exponent – ישות המתארת חזקה.
8. 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.

בידקו את עצמכם ע"י קוד הלקוח הבא:

```
public class Client
{
    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()); // using toString()
        System.out.println(e2 + " = " + e2.eval()); // using toString()
    }
}
```

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)
- שערוך 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.

הצג תרשים מחלקות המתאר את היחס בין הישויות שהגדרת (מכונה גם עץ הורשה או היררכית מחלקות).

מה להגיש ?

(1) קבצים: הקובץ jar ex5. יכיל את הקבצים הבאים:

il.ac.tau.cs.<your user name>.ex5.q1

- DelegatingFixedCapacityQueue.java
- ExtendedFixedCapacityQueue.java
- FixedCapacityQueue.java
- TestDelegatingFixedCapacityQueue.java
- TestExtendedFixedCapacityQueue.java

il.ac.tau.cs.<your user name>.ex5.q3

- Expression.java
- Literal.java
- BinaryOp.java
- TrenaryOp.java
- Sum.java
- Product.java
- Exponent.java
- CondExp.java

הקובץ ימוקם בתיקייה ~/software1 עם הרשאות גישה מתאימות כפי שמפורט במסמך הנחיות ההגשה. יש להקפיד על הפרדת הקבצים לחבילות כמפורט לעיל.

(2) תדפיסים:

פתרון שאלה 1 צריך להכיל את תדפיסי כל הקבצים שהוגשו בקובץ jar בתת החבילה q1. עבור שתי המחלקות היורשות יש לכתוב חוזה ולספק API (תדפיסי Javadoc). קובצי ה-Java עם התחילית Test הם קובצי בדיקת יחידה (unit test) שיכילו 3 מתודות בדיקה test1, test2, test3 שיבדקו 3 תסריטים לא מנוונים של שימוש במחלקה הנבדקת. מתודות אלו יזרקו שגיאה מטיפוס FixedCapacityQueueError במקרה של שגיאה. מחלקות הבדיקה (ללא הבדיקות עצמן) ומחלקת השגיאה זמינות באתר הקורס.

פתרון שאלה 2 צריך להכיל תדפיסי פלט Javadoc עבור כל ישויות התוכנה שיופיעו במערכת שתכנתם (מחלקות ומנשקים). אין צורך לספק את קובצי ה java עצמם. יש לצרף תרשים מחלקות שיתאר בעזרת מלבנים וחיצים את הקשרים בין הישויות השונות. יש לצרף לתרשים דיון קצר שיסביר מדוע נבחרו הקשרים כפי שנבחרו. כמו כן, אין צורך בחוזה מימוש עבור המתודות השונות.

פתרון שאלה 3 צריך להכיל את תדפיסי כל הקבצים שהוגשו בקובץ jar בתת החבילה q3 וכן תרשים מחלקות (מלבנים וחיצים) המתאר את הקשרים בין הישויות השונות. אין צורך בתדפיסי Javadoc עבור שאלה זו.

ליצירת תרשימי המחלקות העזרו במדריך <http://www.classdraw.com/Help.htm> המסכם את סוגי היחסים האפשריים בין מחלקות ומנשקים.