מעבדה 3. נושא: רשימה מקושרת דו-כיוונית

תאריך הגשה: חמישי 30/05/2024 בשעה 23:00 (בזוגות)

יש לקרוא היטב לפני תחילת העבודה !

שימו לב: מעבדה זו היא חובה להגשה,
מאחר ונשתמש בפתרונה במעבדות הבאות.

מבוא:

במעבדה זו נכתוב ממשקים ומחלקות גנריות, וכן נבדוק אותם באמצעות JUnit.
List הוא ממשק גנרי התומך בפעולות המתבצעות על רצף של איברים עם מושג של "איבר נוכחי(סמן)".
במעבדה זו נעסוק במימוש של List ע"י רשימה מקושרת דו-כיוונית.
שימו לב כי אין להשתמש בספריית אוספים כלשהי ובפרט בספריית האוספים התקנית ב java.util.
עליכם להוריד את הקבצים, ולהשלים את הנדרש על פי ההנחיות בפרק "סדר העבודה ופרטים טכניים" בהמשך.

תיאור:

במעבדה זו נעסוק במימוש של List ע"י רשימה מקושרת דו-כיוונית.
המנשק List<T> נתון לכם בקבצי המטלה, והמתודות שלו כדלהלן:
שימו לב כי T הוא שם של טיפוס כלשהו (הניחו כי הטיפוס הוא מחלקה).

void insert (T newElement)
Precondition: newElement is not null.
Postcondition: Inserts newElement into the list. If the list is not empty, then inserts newElement after the cursor. Otherwise, inserts newElement as the first (and only) element in the list. In either case, moves the cursor to newElement.

T remove ()
Precondition: NONE
Postcondition: Removes the element marked by the cursor from the list. If the resulting list is not empty, then moves the cursor to the element that followed the deleted element. If the deleted element was at the end of the list, then moves the cursor to the beginning of the list. Returns the deleted element, or null if the list was empty.

T remove (T element)
Precondition: NONE
Postcondition: Removes element from the list. Moves the cursor to the element that followed the deleted element. If the deleted element was at the end of the list, then moves the cursor to the beginning of the list. Returns the deleted element, or null if it did not exist in the list.
If this element appears several times, removes the first occurrence of it.

void clear ( )
Precondition: None.
Postcondition: Removes all the elements in a list.

void replace (T newElement)
Precondition: List is not empty and newElement is not null.
Postcondition: Replaces the element marked by the cursor with newElement. The cursor remains at newElement

boolean isEmpty ( )

Precondition: *None.*
Postcondition: *Returns true if the list is empty. Otherwise, returns false.*

boolean goToBeginning ( )
Precondition: *None.*
Postcondition: *If the list is not empty, then moves the cursor to the beginning of the list, and return true. Otherwise, returns false.*

boolean goToEnd ( )
Precondition: *None.*
Postcondition: *If the list is not empty, then moves the cursor to the end of the list. Otherwise, returns false.*

T getNext ( )
Precondition: *NONE*
Postcondition: *If the cursor is not at the end of a list, then* ***moves the cursor*** *to the next element and returns it. Otherwise, returns null.*

T getPrev ( )
Precondition: *NONE.*
Postcondition: *If the cursor is not at the beginning of a list, then* ***moves the cursor*** *to the preceding element and returns it. Otherwise, returns null.*

boolean hasNext ( )
Precondition: *NONE*
Postcondition: *If the cursor is not at the end of the list then returns true. Otherwise, returns false.*
If the list is empty, returns false.

boolean hasPrev ( )
Precondition: *NONE.*
Postcondition: *If the cursor is not at the beginning of the list then returns true. Otherwise, returns false.*
If the list is empty, returns false.

T getCursor ( )
Precondition: *NONE.*
Postcondition: *Returns the element marked by the cursor or null if the list is empty.*

1) כתבו מחלקה גנרית DLinkedList<T> העונה על הדרישות הבאות:

a. מממשת את הממשק הנ"ל. יש לממש באופן היעיל ביותר האפשרי מבחינת סיבוכיות זמן ריצה.

שימו לב: יש לבדוק במימוש אם תנאי ה-precondition מתקיימים, ואם לא יש לזרוק חריגה.

b. בעלת בנאי אחד ללא פרמטרים

c. מממשת toString משלה

d. משתמשת במחלקה פרטית בשם DNode כדלקמן (עליכם להגדיר את המחלקה DNode).

אין להוסיף עוד מתודות מעבר למתודות הנתונות במנשק שלהלן.

שימו לב שניתן להשתמש במחלקה DNode רק בתוך המחלקה DLinkedList מאחר ו-DNode מוגדרת כפרטית. אמנם, השדות של DNode חשופים במחלקה DLinkedList על אף שהם מוגדרים פרטיים, בכל זאת, מחוץ למחלקה DNode הקפידו לא לגשת ישירות לשדות של DNode

```java
private class DNode {
    private T element; // element in the list
    private DNode next; // reference to the next element
    private DNode prev; // reference to the previous element

    public DNode(T element) {
        this.element = element;
    }

    public T getElement() {
        return element;
    }

    public void setNext(DNode next) {
        this.next = next;
    }

    public DNode getNext() {
        return next;
    }

    public void setPrev(DNode prev) {
        this.prev = prev;
    }

    public DNode getPrev() {
        return prev;
    }
}
```

2) השלימו את כתיבת המחלקה ה**אבסטרקטית** ListTest<T> המשתמשת ב- Junit4,

ומכילה שיטות לבדיקת עצמים מסוג DLinkedList<T>.

שימו לב כי עליכם **להוסיף** מתודת בדיקה אחת בשם testHasNextAndPrev.

כותרת המתודה הזו נתונה לכם בסוף מחלקת הבדיקות (עם אנוטציה @Test).

ניתן להוסיף עוד בדיקות כרצונכם - בסוף הקובץ.

המטרה היא לבדוק את מימוש המתודות hasNext ו-hasPrev גם עבור מקרי הקצה.

אל תשנו את הקוד במתודות הנתונות, אלא רק הוסיפו מתודות בדיקה נוספות.

למשל: במחלקה נתון לכם שדה private List<T> dList המאותחל במתודה setUp להצביע על מופע של המחלקה DLinkedList<T>, אל תגעו בחלק הזה (אמנם זה שקול ל- dList=new>() DLinkedList<T , ובכל זאת, אל תגעו באתחול זה, בכדי לאפשר הרצה של הבדיקות בבודק האוטומטי)

במחלקה זו יש מתודה **אבסטרקטית** getParameterInstance שתמומש במחלקות היורשות ממחלקה זו. מטרת המתודה ליצור אובייקט חדש של המחלקה T.

כמובן, תוכלו להשתמש במתודה אבסטרקטית זו בכדי לכתוב בדיקות במחלקה ListTest<T>

3) לאחר השלמת שלב 2. התבוננו במחלקה ListTestObject הנתונה לכם.

מחלקה זו נתונה לכם. היא יורשת מ-ListTest<Object>, ומכילה בדיקות למחלקה DLinkedList<Object>.

הריצו את מחלקת הבדיקה הזו וודאו כי כל טסטים עוברים בהצלחה.

4) השלימו את מחלקת הבדיקה ListTestInteger היורשת מ-ListTest<Integer>

ומכילה בדיקות למחלקה DLinkedList<Integer>. יש להשלים מתודה אחת בלבד.

הריצו את מחלקת הבדיקה הזו וודאו כי כל טסטים עוברים בהצלחה.

שימו לב, כל הטסטים כתובים במחלקה הגנרית ListTest<T> תוך שימוש ביצירת מופעים ע״י המתודה getParameterInstance.

המחלקות ListTestObject ו-ListTestInteger ממשות רק את getParameterInstance על פי התנאים הנדרשים.

באופן זה נעשה שימוש במחלקה האבסטרקטית ListTest<T> בצורה מיטבית, בכדי למנוע שכפולי קוד, ולתפוס את הבדיקות המשותפות לכל LinkedList<T> (ובפרט עבור ListTestObject ו -ListTestInteger).

סדר העבודה ופרטים טכניים

- שליפת הפרוייקט DS-Lab03-DLinkedList מתוך GITHUB:
  - אם אין לכם גישה לפרויקט שהורדתם מ GITHUB במעבדה הקודמת יש לבצע שליפה מחדש לפי ההוראות במעבדה הראשונה.
    https://github.com/michalHorovitz/DSLab2024Public
  - אם יש לכם גישה לפרויקט שהורדתם מ GITHUB במעבדה הראשונה אז בצעו:
    - קליק על שם הפרויקט.
    - עכבר ימני
    - Team-->Pull
    - File-->Import->Git->Projects From Git->Existing Local Repository

אם אתם עובדים ב VDI, מומלץ לשנות את המיקום המוצע לפרויקט בתיקייה כלשהי בכונן H.

- הוסיפו את המחלקה DLinkedList<T>.
- השלימו את המחלקה ListTest<T>. (זכרו: אין לשנות במימוש המחלקה אלא רק היכן שרשום TODO, ניתן להוסיף קוד שלכם על פי ההנחיות).
- וודאו כי הרצת ListTestObject עוברת בהצלחה.
- השלימו את המחלקה ListTestInteger וודאו כי הרצת מחלקה זו עוברת בהצלחה.
- יש להגיש את קבצי ה-java.

פורמט קובץ ההגשה ובדיקתו:

פורמט: יש להגיש קובץ ZIP בשם

51_lab03_123456789_987654321.zip

(כמובן, יש להחליף את המספרים עם מספרי ת.ז. של המגישים).

על הקובץ להכיל את כל קבצי ה JAVA שכתבתם. שימו לב: הקובץ לא יכיל את התיקיה שבה הקבצים נמצאים, רק את הקבצים עצמם (אם לא ברור מה ההבדל, ראו סרטון הדגמה מטה).

בדיקת קובץ ההגשה: בדקו את הקובץ שיצרתם בתוכנת הבדיקה בקישור:

https://csweb.telhai.ac.il/

ראו סרטון הדגמה של השימוש בתוכנת הבדיקה.

הסבר על תוצאות הבדיקה האוטומטית:

במעבדה זו, יש מספר בדיקות. סך הציון בבודק האוטומטי הוא **96**. כאשר 4 נקודות נוספות יינתנו באופן ידני עבור הבדיקה שהוספתם בשלב 2.

כמובן, שכל התרגיל יבדק גם ידנית לפי הצורך, כך שהציון בבודק האוטומטי אינו סופי, אך בהחלט משקף את המציאות.

תרגיל שהוגש בפורמט נכון ועבר קומפילציה בבודק האוטומטי יקבל על כך 40 נק׳.

יש שלשה סטים של בדיקות - כולם ב-junit.
הטסטים (מלבד אחד) נתונים לכם במחלקה ListTest.

1. 12 tests שבודקים את המחלקה DLinkedList<T> שמימשתם, עבור T=Integer.
   סה״כ 20 נק׳ עבור סט זה של בדיקות.
   בסט זה נבדקת גם מימושה של המתודה getParameterInstance ב-ListTestInteger.

2. 11 tests שבודקים את המחלקה DLinkedList<T> שמימשתם, עבור T=String.
   סה״כ 18 נק׳ עבור סט זה של בדיקות.

3. 11 tests שבודקים את המחלקה DLinkedList<T> שמימשתם, עבור T=Object.
   סה״כ 18 נק׳ עבור סט זה של בדיקות.

סה״כ 96 נק׳ בבודק האוטומטי, ועוד 4 נק׳ יוספו בבדיקה ידנית עבור משימת כתיבת הטסט.

חשוב !!!
בדיקת ההגשות תבוצע ברובה ע"י תוכנית הבדיקה האוטומטית הנ"ל. תוצאת הבדיקה תהייה בעיקרון זהה לתוצאת הבדיקה הנ"ל שאתם אמורים לערוך בעצמכם . כלומר, אם ביצעתם את הבדיקה באתר החוג, לא תקבלו הפתעות בדיעבד. אחרת, ייתכן שתרגיל שעבדתם עליו קשה ייפסל בגלל פורמט הגשה שגוי וכו'. דבר שהיה ניתן לתקנו בקלות אם הייתם מבצעים את הבדיקה. היות ואין הפתעות בדיעבד, לא תינתן אפשרות של תיקונים, הגשות חוזרות וכד'.

הגשה שלא מגיעה לשלב הקומפילציה תקבל ציון 0.
הגשה שלא מתקמפלת תקבל ציון נמוך מ- 40 לפי סוג הבעיה.
הגשה שמתקמפלת תקבל ציון 40 ומעלה בהתאם לתוצאות הריצה, ותוצאת הבדיקה הידנית של הקוד (חוץ ממקרה של העתקה).
תכנית הבדיקה האוטומטית מכילה תוכנה חכמה המגלה העתקות. מקרים של העתקות יטופלו בחומרה