עבודה עצמית – שבוע 6

הורידו את הקוד הנמצא בקישור הבא:
http://courses.cs.tau.ac.il/software1/2021a/lectures/tutorials/resources/class6.zip
ופתחו פרוייקט חדש ב eclipse עבורו.

אנחנו ממשיכים עם המחלקה MyList אשר מומשה במהלך ההרצאה. בהרצאה ראינו שתי אופציות למימוש Cell כמחלקה פנימית. הקוד שהורדתם מכיל מימוש של אחת האלטרנטיבות שהוצגו – המחלקה Cell כמחלקה פנימית סטטית.

עצרנו את השיעור בשאלה: כיצד ניתן לאפשר למשתמשת לעבור על הרשימה המקושרת מבלי לחשוף את המבנה הפנימי שלה, ומבלי להגביל את המשתמשת לשירותים כמו printList.

ראשי, ניזכר כיצד אנחנו עוברים על מבני נתונים מוכרים? ניקח כדוגמא מערך של תוים ונעבור עליו בלולאת for:

```
char[] letters = {'a','b','c','d','e','f'};

void printLetters() {
      System.out.print("Letters: ");

      for (int i=0 ; i < letters.length ; i++) {
            System.out.print(letters[i] + " ");
      }

      System.out.println();
}
```

בלולאת ה for אנחנו רואים ארבעה אלמנטים הנחוצים במעבר על המערך:
אתחול משתנה עזר (צהוב)
בדיקת התנאי להמשך הלולאה (ירוק)
קידום משתנה העזר (תכלת)
שימוש במשתנה העזר כדי להגיע לפריט המבוקש במערך (ורוד).

באופן דומה, הקוד שמבצע מעבר בלולאה על הרשימה המקושרת בתוך המימוש של printList מכיל את אותם האלמנטים:

```
public void printList() {
      System.out.print("List: ");
      for (Cell<T> y = head; y != null; y = y.next())
            System.out.print(y.cont() + " ");
      System.out.println();
}
```

לולאה זו לא יכולה להתבצע ע"י הלקוחה שלנו כיוון שהיא כלל לא מכירה את המחלקה Cell והשדה head, ומצד שני, השירות printList אינו מספיק לכל הצרכים האפשריים של הלקוחה, ולכן נשתמש במושג שנקרא Iterator.

ה Iterator (אצן או סודר בעברית) מאפשר לנו לעבור בצורה סדרתית על איברי מבנה נתונים כלשהו מבלי להחשף למבנה הפנימי שלו. למעשה, זוהי הפשטה של מעבר על מבנה הנתונים שכן אנחנו

מוגבלים למספר מצומצם של פעולות, מבלי לחשוף החוצה את המימוש של אותו מבנה נתונים. זוהי תבנית עיצוב (design pattern) שרלוונטית לשפות תכנות שונות, בין השאר ניתן לפגוש אותה גם ב Python.

ב Java מוגדר המנשק Iterator<S> שמגדיר את השירותים האבסטרקטיים הבאים:

א. hasNext – השירות בודק אם קיימים איברים נוספים למבנה הנתונים עליהם טרם עברנו. כאשר הוא מחזיר false המשמעות היא שסיימנו לעבור על כל האיברים.

ב. next – השירות מבצע שתי פעולות:

a. מבצע קידום לאיבר הבא.

b. מחזיר את האיבר הבא, שהוא איבר מטיפוס S (הטיפוס הגנרי של המנשק)

(דומה לפעולת i++ שגם היא מבצעת שתי פעולות בו זמנית – מגדילה את i ב 1 ומחזירה את ערכו).

אם נשים לב, שתי הפעולות, hasNext ו next מכסות בעצם 3 אלמנטים שראינו בלולאות. פעולת ה hasNext היא הפעולה הצבועה בירוק – בדיקה אם ניתן להמשיך, והפעולה next מאחדת שני אלמנטים: קידום משתנה העזר (תכלת) וגישה לפריט המבוקש במבנה הנתונים (ורוד).

בנוסף לשתי פונקציות אלה, שהן אבסטרקטיות, המנשק מגדיר פונקציות נוספות להן קיים מימוש דיפולטי. את השירות remove (מחיקת האיבר הנוכחי ממבנה הנתונים) תראו בהמשך הסמסטר בתרגול.

השימוש ב Iterator יעשה באופן הבא:

```
for (Iterator iter = collection.iterator(); iter.hasNext( ); ) {
      System.out.println(iter.next());
}
```

בהנתן אוסף כלשהו (collection בדוגמא זו), נבקש ממנו את האיטרטור שלו. בדוגמא זו, זה נעשה באמצעות השירות ()iterator. זהו בעצם שלב האתחול של משתנה העזר.
נעזר ב hasNext על מנת לראות אם ניתן להמשיך את הלולאה.
בגוף לולאת ה for לא נבצע קידום (שימו לב שהחלק האחרון של מבנה ה for הוא ריק). מדוע? כי הקידום יתבצע כשנבצע ()next בגוף הלולאה (הפעולה צבועה באפור והיא בעצם משלבת את שתי הפעולות שקודם צבענו בתכלת ובורוד).

להלן מימוש אפשרי לאיטרטור של המחלקה MyList:

```
private static class MyListIterator<S> implements Iterator<S> {
      private Cell<S> curr;

      public MyListIterator(Cell<S> cell) {   this.curr = cell;   }

      public boolean hasNext() {  return curr != null;  }

      public S next() {
            S result = curr.cont();
            curr = curr.next();
            return result;
      }
}
```

המימוש שבו בחרנו הוא מחלקה סטטית בתוך MyList (ניתן היה לבחור במימושים אחרים).
שימו לב לפרטים הבאים:

א. המחלקה MyListIterator היא מחלק פנימית בתוך MyList ולכן מכירה את Cell, שגם היא מחלקה פנימית.
ב. המחלקה MyListIterator מחזיקה מצביע ל Cell, ובכל פעם שהיא מבצעת קידום של האיטרטור, אנחנו למעשה מדלגים ל next של ה Cell הנוכחי.
ג. במימוש של next אנחנו רואים שמבוצעות שתי פעולות: שמירת הערך הנוכחי עליו מצביע curr וקידומו לאיבר הבא. זה עקבי עם זה שאנחנו עוצרים את הלולאה רק כש curr == null. אם היינו מממשים את hasNext באמצעות הבדיקה curr.next() == null היינו מפספסים את האיבר האחרון.
שימו לב שיהיו מבני נתונים שבהם קודם נבצע קידום ואז נקבע את הערך שמוחזר, זה תלוי במימוש הפנימי של מבנה הנתונים.

כעת, העתיקו את מימוש המחלקה MyListIterator למחלקה MyList (כמחלקה פנימית), והוסיפו ל MyList את שירות ה main הבא:

```
public static void main(String[] args){
	MyList<Integer> mList = new MyList<Integer>(1,2,3,4,5);
	Iterator<Integer> mListIterator =
					new MyListIterator<Integer>(mList.head);
	while( mListIterator.hasNext()){
		System.out.println(mListIterator.next());
	}
}
```

על מנת שהקוד יתקמפל, עליהם לבצע import למחלקה Iterator מתוך java.util (ה eclipse יציע זאת בעצמו).

הריצו את הקוד ו-ודאו שכל איברי mList אכן מודפסים. שימו לב לנקודות הבאות:

א. כאשר אנחנו מייצרים את mListIterator הוא צריך לקבל בבנאי שלו את ה Cell שמייצג את תחילת הרשימה, ולכן אנחנו שולחים לו את mList.head שתמיד מצביע לאיבר הראשון.
ב. בחרנו להמיר את לולאת ה for ללולאת ה while כיוון שלב אתחול משתנה העזר בוצע בנפרד, ואין צורך לבצע פעולה נפרדת לקידום (מתבצע גם ככה באמצעות הפקודה ()next).
ג. המשתנה mListIterator הוא מטיפוס סטטי <Iterator<Integer, והטיפוס הדינאמי שלו הוא <MyListIterator<Integer. ראינו כבר בשיעור על מנשקים שטיפוס סטטי יכול להיות מנשק, והטיפוס הדינאמי הוא מחלקה המממשת מנשק זה.

במימוש הנוכחי יש בעיה – מחוץ למחלקה MyList לא ניתן יהיה לייצר את האיטרטור הזה, כיוון שכפרמטר הוא צריך לקבל את mList.head (שהוא שדה private). בנוסף, הגדרנו שהניראות של המחלקה MyListIterator היא גם כן private.
לכן, הפתרון שנשתמש בו הוא מימוש שירות בתוך MyList אשר ייצר את האיטרטור. מימוש אפשרי:

```
public Iterator<T> getIterator(){
	return new MyListIterator<T>(this.head);
}
```

הוסיפו את השירות הזה למחלקה MyList, וכעת ניתן יהיה להמיר את השורה השניה ב main בשורה:

```
Iterator<Integer> mListIterator = mList.getIterator();
```

כעת, הקוד שמופיע ב main יכול לרוץ מכל מחלקה, ואין בעית ניראות .שימו לב שב main אנחנו לא עושים שימוש בשום שדה פנימי של MyList, וכמו כן, אין שימוש מפורש במחלקה MyListIterator.

האם ניתן לייצר איטרטורים נוספים? כמובן!
נוסיף איטרטור אשר עובר על הרשימה המקושרת בדילוגים.

```
private static class MySkipIterator<S> implements Iterator<S> {
	private Cell<S> curr;

	public MySkipIterator(Cell<S> cell) {
		this.curr = cell;
	}

	public boolean hasNext() {
		return curr != null ;
	}

	public S next() {
		S result = curr.cont();
		curr = curr.next();
		curr = curr == null? null : curr.next();
		return result;
	}
}
```

מימוש זה זהה למעשה למימוש של האיטרטור הקודם, פרט למימוש של next. כאשר אנחנו רוצים להתקדם לאיבר הבא, אנחנו רוצים לדלג על איבר, ולכן צריך לבצע שתי פעולות next על curr. פעולת ה next הראשונה היא בטוחה, אנחנו יודעים ש curr אינו null (אם עושים שימוש נכון באיטרטור, לא מבצעים קריאה ל next לפני קריאה ל hasNext). הקריאה השניה ל next (שורה לפני אחרונה) היא לא בטוחה, יכול להיות ש curr אינו null, אבל אחרי שביצענו curr = curr.next() הוא כן מצביע ל null. פה נשתמש באופרטור הטרנרי (ראינו בתחילת הקורס), ונתקדם רק אם מותר להתקדם. אם לא, curr יהיה שווה ל null והאיטרציה הזו תהיה למעשה האיטרציה האחרונה.

כעת, נוכל להוסיף פונקציה נוספת ל MyList, בדומה ל getIterator, והיא תחזיר את האיטרטור הזה. באופן דומה, ניתן להוסיף כמה איטרטורים שרוצים.

האם אנחנו חייבים לממש שירות כדוגמת getIterator בשביל להשתמש באיטרטור? כבר ראינו שלא. במידה ויש בעיה של ניראות\חשיפת מימוש פנימי נוח שיש פונקציה כזו, אבל ניתן לכתוב איטרטור גם עבור מבני נתונים שלא מצריכים גישה למבנה הפנימי. למשל, יכולנו לממש SkipIterator עבור מערך.

המחלקה MyArrSkipIterator (מופיעה בקובץ שהורדתם) מממשת איטרטור אשר עובר על מערכים בדילוגים (המחלקה מופיעה בתיקיה שהורדתם). בבנאי של האיטרטור נקבל 2 פרמטרים: המערך, והצעד – גודל הקפיצה. בתוך המחלקה עצמה יש שלושה שדות מופע: המערך עליו אנחנו עוברים,

הצעד, והאינדקס של האיבר הבא אותו נרצה להחזיר. כאשר האינדקס של האיבר הבא חורג מגודל המערך, hasNext תחזיר False.

הריצו את MyArrSkipItertorTester ושנו את הפרמטרים לאתחול האיטרטור על מנת לבדוק את נכונות המימוש.

עד כה ראינו לולאת for ולולאת while עם איטרטורים. ומה עם לולאת for each? זה מרגיש מאוד טבעי, עבור MyList, להיות מסוגלים לכתוב קוד כזה:

```
MyList<Integer> mList = new MyList<Integer>(1,2,3,4,5);
for (int i: mList){
            System.out.println(i);
}
```

אם ננסה להריץ את הקוד הזה במימוש הנוכחי של MyList הוא לא יתקמפל, אבל ניתן לגרום לזה לעבוד. זה פשוט יצריך מאיתנו לממש מנשק נוסף.

ראינו כבר שלכל מבנה נתונים יכולים להיות מממומשים כמה איטרטורים (או אף איטרטור, כמובן). חלקם ממומשים בתוך המחלקה, וחלק לא מוכרים למחלקה כלל (כמו האיטרטור שעובר על מערך בדילוגים). התחביר של לולאת ה for each מכתיב עבודה עם מבנה הנתונים בלבד (mList) ולא עם איטרטור, ולכן מה שעשינו ב for/while לא יעבוד. איכשהו mList צריכה לדעת בעצמה באיזה איטרטור להשתמש בלולאת for each. ומי בכלל הבטיח שמומש לה איטרטור? מה יקרה אם נכתוב לולאת for each עבור מבנה נתונים שאין לו אף איטרטור?

לצורך כך קיים מנשק נוסף: Iterable. מנשק זה מתאר מבנה נתונים כלשהו, ומשמעותו: מבנה נתונים זה תומך במעבר איטרטיבי, ובפרט, ניתן לעבור עליו גם בלולאת for each.
המנשק מגדיר פונקציה יחידה: iterator() שלמעשה מחזירה אובייקט מסוג Iterator.

אם נחזור לקוד של MyList, הוספנו לו את השירות getIterator שיחזיר את האיטרטור שעובר עליו באופן סדרתי. זוהי בדיוק הפונקציה שמגדיר המנשק Iterable, אלא שהשם שלה צריך להשתנות.

שנו את ההגדרה של MyList להגדרה הבאה:

```
public class MyList<T> implements Iterable<T>
```

כעת, תצטרכו לממש את הפונקציה iterator() שהמנשק מגדיר. העתיקו את המימוש מהשירות getIterator וודאו שכעת ניתן להריץ את לולאת ה for each שכתבנו בתחילת עמוד זה.

אגב, השירות iterator יכול היה לחזיר גם את MySkipIterator שמומש ב MyList. אם נחליף את המימושים, לולאת ה for each תדלג על כל איבר שני ב MyList .