

תוכנה 1 בשפת Java אצנים ואוספים (חלק א')

תרגול מספר 7
אורנית דרור ואוהד ברזילי

איטרטור (סודר? אצן? סורק?)

- איטרטור הוא הפשטה של מעבר בסדר מוגדר מראש על מבנה נתונים כלשהו
- כדי לבצע פעולה ישירה על מבנה נתונים, יש לדעת כיצד הוא מיוצג
- גישה בעזרת איטרטור למבנה הנתונים מאפשרת למשתמש לסרוק מבנה נתונים ללא צורך להכיר את המבנה הפנימי שלו



הדפסת מערך (אינדקסים)

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

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

גישה בעזרת
משתנה העזר
לנתון עצמו

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

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

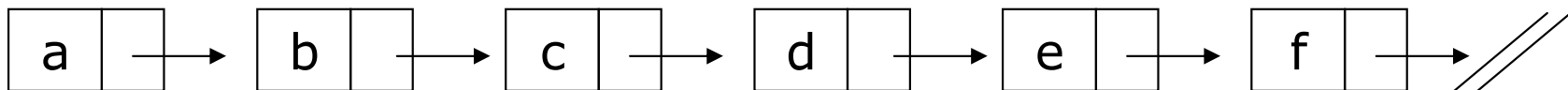
הגדרת
משתנה עזר
ואתחולו

בדיקה:
האם גלשנו

קידום משתנה העזר
(מעבר לאיבר הבא)

הדפסת רשימה מקושרת

```
public class Cell<T> {  
    private T content;  
    private Cell<T> next;  
  
    public Cell (T content, Cell<T> next) {  
        this.content = content;  
        this.next = next;  
    }  
    public T getContent () {  
        return content;  
    }  
    public Cell<T> getNext () {  
        return next;  
    }  
    public void setNext (Cell<T> next) {  
        this.next = next;  
    }  
}
```



הדפסת רשימה מקושרת - המשך

...

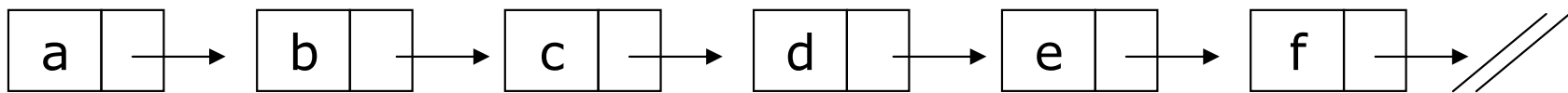
```
public void printList() {  
    System.out.print("List: ");  
  
    for (Cell<T> y = this ; y != null ; y = y.getNext()) {  
        System.out.print(y.getContent() + " ");  
  
        System.out.println();  
    }  
}
```

הגדרת
משתנה עזר
ואתחולו

בדיקה:
האם גלשנו

גישה בעזרת
משתנה העזר לנתון
עצמו

קידום משתנה העזר
(מעבר לאיבר הבא)



הכרות אינטימית עם מבנה הנתונים

2 הדוגמאות הקודמות חושפות ידע מוקדם שיש לכותבת פונקצית ההדפסה על מבנה הנתונים:

- היא יודעת איפה הוא מתחיל ואיפה הוא נגמר
- היא מכירה את מבנה הטיפוס שבעזרתו ניתן לקבל את המידע השמור במצביע
- היא יודעת איך לעבור מאיבר לאיבר שאחריו

בדוגמת הרשימה המקושרת כותבת המחלקה Cell (הספקית) היא זו שכתבה את מתודת ההדפסה

■ זה אינו מצב רצוי - זהו רק מקרה פרטי של פעולה אחת מני רבות של **שלקוחות** עשויים לרצות לבצע על מחלקה. על המחלקה לספק **כלים** ללקוחותיה לבצע פעולות כאלו בעצמם

האיטרטור

- איטרטור הוא בעצם **מנשק** (interface) המגדיר פעולות יסודיות שבעזרתן ניתן לבצע מגוון רחב של פעולות על אוספים
- ב Java טיפוס יקרא Iterator אם ניתן לבצע עליו 4 פעולות:
 - בדיקה האם גלשנו (`hasNext ()`)
 - קידום (`next ()`)
 - גישה לנתון עצמו (`next ()`)
 - הסרה של נתון (`remove ()`) – אופציונלי
- כן, זה נורא! `next ()` היא גם פקודה וגם שאילתה
- ממש כשם שמימושים מסוימים של `pop ()` על מחסנית גם מסירים את האיבר העליון וגם מחזירים אותו
- בשפות אחרות (`C++` או Eiffel):
 - יש הפרדה בין קידום משתנה העזר והגישה לנתון
 - `remove ()` אינה חלק משרותי איטרטור (וכך גם אנו סבורים)

אלגוריתם כללי להדפסת אוסף

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

גישה בעזרת
משתנה העזר לנתון
וקידומו לאיבר הבא

■ מבנה הנתונים עצמו אחראי לספק ללקוח איטריטור תיקני (עצם ממחלקה שמממשת את ממשק Iterator) המאותחל לתחילת מבנה הנתונים

■ נרצה שהמחלקה ce11 תספק ללקוחותיה את האפשרות לסרוק את האיברים ברשימה ממנה ואילך. לשם כך עליה לספק להם Iterator

הגדרת
משתנה עזר
ואתחולו

בדיקה:
האם גלשנו

תקני CellIterator

```
class CellIterator<S> implements Iterator<S> {
    public CellIterator(Cell<S> cell) {
        this.curr = cell;
    }

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

    public S next() {
        S result = curr.getContent();
        curr = curr.getNext();
        return result;
    }

    public void remove() {} // must be implemented

    private Cell<S> curr;
}
```

Cell מספקת איטרטור ללקוחותיה

```
public class Cell<T> implements Iterable<T> {  
    //...  
    public Iterator<T> iterator() {  
        return new CellIterator<T>(this);  
    }  
}
```

■ מחלקות המממשות את המתודה `iterator()` בעצם מממשות את הממשק `Iterable<T>` המכיל מתודה זו בלבד

■ הצימוד בין `Cell` ו-`CellIterator` חזק. בהמשך הקורס, כאשר נלמד מחלקות פנימיות נממש את האיטרטור כמחלקה פנימית של האוסף שעליו הוא פועל

■ כעת הלקוח יכול לבצע פעולות על כל אברי הרשימה בלי לדעת מהו המבנה הפנימי שלה

printSquares

```
public void printSquares (Cell<Integer> head) {  
    for (Iterator<Integer> iter = head.iterator();  
        iter.hasNext();) {  
        int i = iter.next();  
        System.out.println(i*i);  
    }  
}
```

Autoboxing

What is the output for:

```
System.out.println(iter.next()*iter.next());
```

(שמרו לכן על הפרדה בין פקודות לשאיתות)

- הלקוח מדפיס את ריבועי אברי הרשימה בלי להשתמש בעובדה שזו אכן רשימה
- בהמשך נראה שטיפוס הארגומנט `Cell<Integer>` יכול להיות מוחלף בשם הממשק `Collection` (אוסף נתונים כלשהו), ואז הלקוח לא ידע אפילו את שמו של טיפוס מבנה הנתונים

for/in (foreach)

לולאת for שמבצעת את אותה פעולה על כל אברי אוסף נתונים כלשהו כה שכיחה, עד שב Java 5.0 הוסיפו אותה לשפה בתחביר מיוחד (for/in) הקוד מהשקף הקודם שקול לקוד הבא:

```
public void printSquares (Collection<Integer> head) {  
    for (int i : head)  
        System.out.println(i*i);  
}
```

יש לקרוא זאת כך:

"לכל איבר i מטיפוס int שבאוסף הנתונים ...head"

אוסף הנתונים head חייב לממש את הממשק `Iterable`

for/in (foreach)

ניתן לעבוד עם מערכים כטיפוס `:Iterable` ■

```
int[] arr = {6,5,4,3,2,1};  
for (int i : arr) {  
    System.out.println(i*i);  
}
```