

## 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



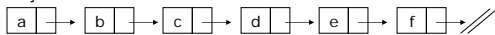
## תכנות מתקדם בשפת Java אצנים ואוספים (collections and iterators)

אורנית דרור ואוהד ברזילי  
אוניברסיטת תל אביב

## 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 מערך (אינדקסים)

```
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();
}
```



## הכרות אינטימית עם מבנה הנתונים

- 2 הדוגמאות הקודמות חושפות ידע מוקדם שיש לכותבת פונקציית 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 שלקוחות עשויים לרצות לבצע על מחלקה. על המחלקה לספק כלים ללקוחותיה לבצע פעולות כאלו בעצמם

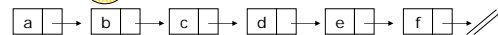
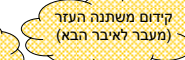
## הדפסת רשימה מקושרת - המשך

```
...

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();
}
```



## האיטרטור

- כן, זה נראה! `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)

## האיטרטור

- איטרטור הוא בעצם **ממשק** (`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

## CellIterator תקיני

```
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;
}
```

## 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 איטרטור תיקני (עצם ממחלקה שממשת את ממשק `Iterator`) המאוחל לתחילת מבנה הנתונים
- נרצה שהמחלקה `Cell` תספק ללקוחותיה את האפשרות לסרוק את האיברים ברשימה ממנה ואילך. לשם כך עליה לספק להם `Iterator`

הגדרת  
משתנה עזר  
ואתחולו

בדיקה  
האם גלשנו

## 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 לשאילתות)

- 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

## 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 במחלקה פנימית של האוסף שעליו הוא פועל
- כעת הלקוח יכול לבצע פעולות על כל אברי הרשימה בלי לדעת מהו המבנה הפנימי שלה

## 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);
}
```

## for/in (foreach)

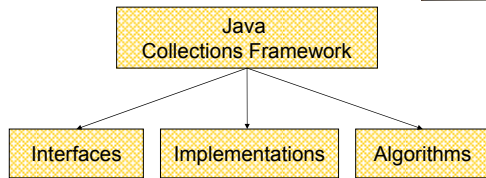
- לולאת for שמבצעת את אותה פעולה על כל אברי אוסף נתונים כלשהו כה שכיחה, עד שב Java 5.0 הוסיפו אותה לשפה בתחביר מיוחד (for/in) הקוד מהשקף הקודם שקול לקוד הבא:

```
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

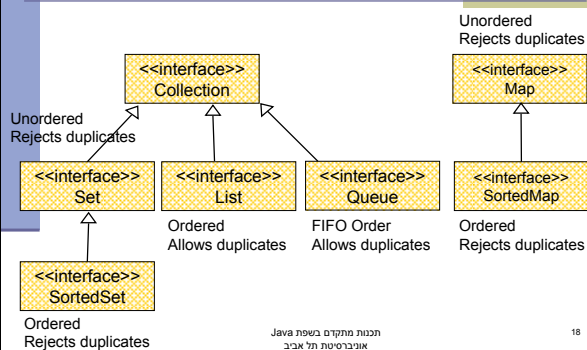
## Java Collections Framework

- Collection: a group of elements
- Interface Based Design:



## Java Collections

## Collection Interfaces



## Online Resources

- Java 5 API Specification:  
<http://java.sun.com/j2se/1.5.0/docs/api/index.html>
- Sun Tutorial:  
<http://java.sun.com/docs/books/tutorial/collections/>

# Collection Implementations

- Class Name Convention: <Data structure> <Interface>

General Purpose Implementations		Data Structures			
		Hash Table	Resizable Array	Balanced Tree	Linked List
Interfaces	Set	HashSet		TreeSet (SortedSet)	
	Queue				LinkedList
	List		ArrayList		LinkedList
	Map	HashMap		TreeMap (SortedMap)	

# The Collection Interface

- Doesn't hold primitives (use wrapper classes)
- Since Java5 collections are type-safe:

```
Collection<String> collectionOfStrings = new LinkedList<String>();
Collection<Integer> collectionOfIntegers = new LinkedList<Integer>();

collectionOfStrings.add("Hello");
collectionOfIntegers.add(5);
collectionOfIntegers.add(new Integer(6));

collectionOfStrings.add(7);
collectionOfIntegers.add("world");
```

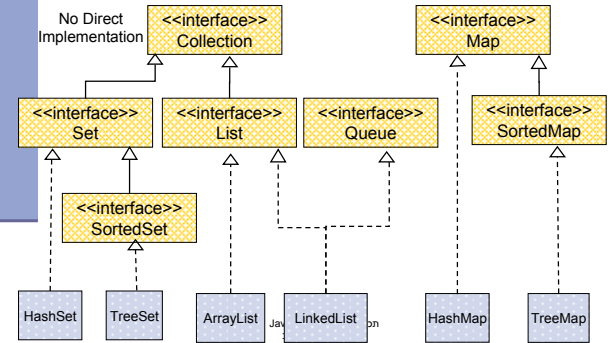
# Best Practice

- Specify an implementation only when a collection is constructed:

```
Set s = new HashSet();
public void foo(HashSet s) {...}
public void foo(Set s) {...}
s.add() invokes HashSet.add()
```

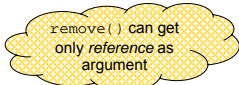


# General Purpose Implementations

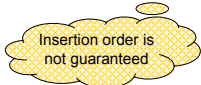


# Set Example

```
Set<Integer> set = new HashSet<Integer>();
set.add(3);
set.add(1);
set.add(new Integer(1));
set.add(new Integer(6));
set.remove(6);
System.out.println(set);
```



Output: [1, 3]

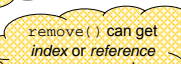
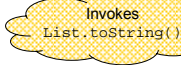
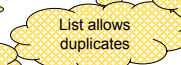
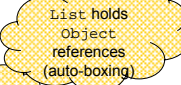
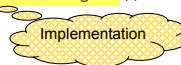


A set does not allow duplicates. it may not contain:

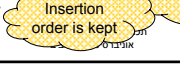
- two references to the same object
- two references to null
- references to two objects a and b such that a.equals(b)

# List Example

```
List<Integer> list = new ArrayList<Integer>();
list.add(3);
list.add(1);
list.add(new Integer(1));
list.add(new Integer(6));
list.remove(list.size()-1);
System.out.println(list);
```



Output: [3, 1, 1]



## Map Example

```
Map<String,String> map = new HashMap<String,String>();
map.put("Dan", "03-9516743");
map.put("Rita", "09-5076452");
map.put("Leo", "08-5530098");
map.put("Rita", "06-8201124");
System.out.println(map);
```

No duplicates

Unordered

Output:

```
{Leo=08-5530098, Dan=03-9516743, Rita=06-8201124}
```

Keys (names)	Values (phone numbers)
Dan	03-9516743
Rita	06-8201124
Leo	08-5530098

26

## Queue Example

```
Queue<Integer> queue = new LinkedList<Integer>();
queue.add(3);
queue.add(1);
queue.add(new Integer(1));
queue.add(new Integer(6));
queue.remove();
System.out.println(queue);
```

Elements are added to the tail of the queue.

remove() may have no argument – head is removed

Output: [1, 1, 6]

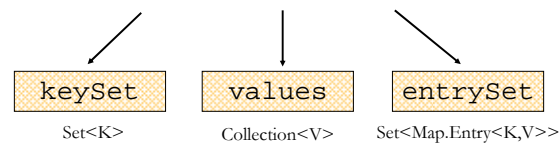
FIFO order

חנות מתקדם בשפת Java  
אנדרסטית תל אביב

25

## Map Collection Views

Three views of a `Map<K, V>` as a collection



חנות מתקדם בשפת Java  
אנדרסטית תל אביב

28

## SortedMap Example

```
SortedMap<String,String> map = new TreeMap<String,String>();
map.put("Dan", "03-9516743");
map.put("Rita", "09-5076452");
map.put("Leo", "08-5530098");
map.put("Rita", "06-8201124");
System.out.println(map);
```

lexicographic order

Output:

```
{Dan=03-9516743, Leo=08-5530098, Rita=06-8201124}
```

Keys (names)	Values (phone numbers)
Dan	03-9516743
Rita	06-8201124
Leo	08-5530098

27

## Iterating Over the Keys of a Map

```
Map<String,String> map = new HashMap<String,String>();
map.put("Dan", "03-9516743");
map.put("Rita", "09-5076452");
map.put("Leo", "08-5530098");
map.put("Rita", "06-8201124");
```

```
for (String key : map.keySet()) {
    System.out.println(key);
}
```

Output: Leo  
Dan  
Rita

חנות מתקדם בשפת Java  
אנדרסטית תל אביב

30

## Iterating Over the Keys of a Map

```
Map<String,String> map = new HashMap<String,String>();
map.put("Dan", "03-9516743");
map.put("Rita", "09-5076452");
map.put("Leo", "08-5530098");
map.put("Rita", "06-8201124");
```

```
for (Iterator<String> iter= map.keySet().iterator(); iter.hasNext();) {
    System.out.println(iter.next());
}
```

Output: Leo  
Dan  
Rita

חנות מתקדם בשפת Java  
אנדרסטית תל אביב

29

## Iterating Over the Key-Value Pairs of a Map

```
Map<String,String> map = new HashMap<String,String>();
map.put("Dan", "03-9516743");
map.put("Rita", "09-5076452");
map.put("Leo", "08-5530098");
map.put("Rita", "06-8201124");

for (Map.Entry entry : map.entrySet()) {
    System.out.println(entry.getKey() + ": " + entry.getValue());
}
```

**Output:**  
Leo: 08-5530098  
Dan: 03-9516743  
Rita: 06-8201124

## Iterating Over the Key-Value Pairs of a Map

```
Map<String,String> map = new HashMap<String,String>();
map.put("Dan", "03-9516743");
map.put("Rita", "09-5076452");
map.put("Leo", "08-5530098");
map.put("Rita", "06-8201124");

for (Iterator<Map.Entry<String,String>> iter = map.entrySet().iterator();
iter.hasNext(); ) {
    Map.Entry<String,String> entry = iter.next();
    System.out.println(entry.getKey() + ": " + entry.getValue());
}
```

**Output:**  
Leo: 08-5530098  
Dan: 03-9516743  
Rita: 06-8201124

## Sorting

```
import java.util.*;

public class Sort {
    public static void main(String args[]) {
        List<String> list = Arrays.asList(args);
        Collections.sort(list);
        System.out.println(list);
    }
}
```

**Arguments:** A C D B  
**Output:** [A, B, C, D]

import the package of List, Collections and Arrays

returns a List-view of its array argument.

lexicographic order

## Collection Algorithms

- Defined in the [Collections](#) class
- Main algorithms:
  - sort
  - binarySearch
  - reverse
  - Shuffle
  - Min
  - max

## Sorting (cont.)

- Sort a List `l` by `Collections.sort(l)`;
- If the list consists of `String` objects it will be sorted in lexicographic order. Why?
- `String` implements `Comparable<String>`:

```
public interface Comparable<T> {
    public int compareTo(T o);
}
```
- Exception when sorting a list whose elements
  - do not implement `Comparable` or
  - are not *mutually comparable*.