Boaz Kantor
Introduction to Computer Science,
Fall semester 2009-2010
IDC Herzliya

Introduction to Data Structures

"Bad programmers worry about the code. Good programmers worry about data structures and their relationships.", - Linus Torvalds, 2006

Agenda

- Background theory, recap
  - Why data structures
  - Abstraction vs. Implementation
  - How to choose our data structure
  - Data structure characteristics
  - Linked list
- Exercise: "Barnes & Noble":
  - Array implementation
  - Linked list implementation
- LinkedList.addElement depiction
- Queue
  - Array implementation
  - Linked list implementation
- Summary

Background theory, recap

Why data structures?

- In order to develop an application or feature, we consider:
  - UI and I/O
  - Algorithm
  - Data handling
- Arrays are a simple data structure: linear, homogenous, direct access.
- These traits are sometimes very limiting and inefficient;
  - We can't define relations between elements, memory allocation is not dynamic, all the cells look the same, etc..
- **We write our own data structures to achieve better flexibility and efficiency**

Abstraction vs. implementation

- Abstraction
  - What we want the data structure to do
- Implementation
  - How it should be done

Abstraction	Implementation
Stack	Array
Stack	Linked list
Stack.pop()	return array[--size];

Stack.pop()	temp = head;
	head = head.next;
	return head;

How to choose our data structure

- We need to choose:
  - Data types and their relationships
  - Data structure
- Choosing data types is an OO task.
- To choose a data structure, ask these questions:
  - How dynamic is the collection?
  - According to what will we want to add/retrieve elements?
  - What are the performance requirements for adding/retrieving elements?

Data structure characteristics

Data structure	Characteristics
Array	Static size, direct access
Linked list	Dynamic size, iterative
Stack	LIFO, access only to last
Queue	FIFO, access only to first
Graph	n:n relationships
Tree	1:n relationships
Hash, dictionary, maps, …	Future

Linked list

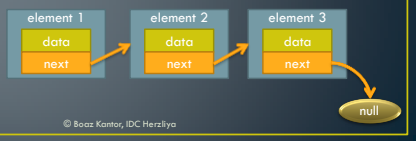
- One of the most basic data structures
- The principal:
  - Keep only a reference to the first element ('head')
  - Each element points to the next one
  - The last element points to null
- Dynamic size:
  - Grows when adding elements, shrinks when removing elements
  - Unlimited number of elements
  - To add an element, "play" with references
- No direct access:
  - To get to an element, start with the head and iterate the entire collection

Linked list abstraction (partial list)

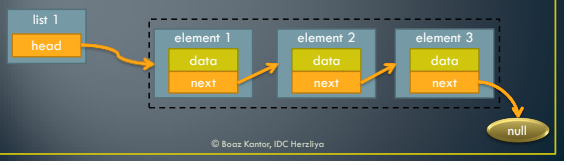
Depending on your encapsulation design, the abstraction may work either with Data or directly with Element.

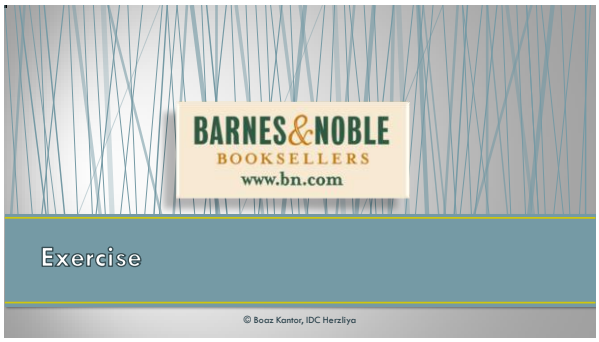
- `void isEmpty()`
- `void insertAtBeginning(Element element)`
- `void append(Element element)`
- `Element getFirst()`
- `Element getLast()`
- `Element getElement(Data data)`
- `void clear()`

```
class ListElement {
        private SomeType data;
        private ListElement next = null;
        // constructors
        // 'data' setters and getters
        // 'next' setters and getters
}
```




```
class LinkedList {
        private ListElement head = null;
        // is the list empty?
        // add element (to beginning, end, or anywhere)
        // remove element
        // get first/last/specific element
}
```





- Exercise: "Barnes & Noble" has asked you to rewrite their ordering system (they're paying a lot).
- They want to provide you with book orders. You need to keep these orders.
- Their storage manager wants to retrieve the orders, oldest order first, so they can process the order and deliver the books.
- Plan:
  - Provide both users with a book orders data type.
  - Provide B&N with an interface for adding orders.
  - Provide the storage manager with an interface for getting the next order.

Step 1, custom data type

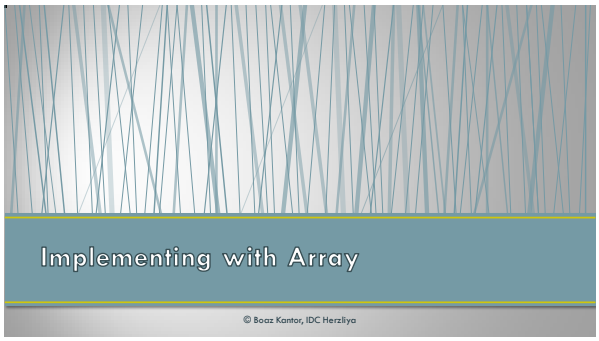
Assume classes Book and Customer (provided by B&N)

```
public class BookOrder {
      private Book book = null;
      private Customer customer = null;
      // constructors, setters and getters
}
```

Step 1.1: implement.

Step 2, SDK for B&N and storage manager

```
public class OrdersCollection {

      // adds an order to the collection
      public void addOrder(BookOrder order)

      // returns and removes the oldest order
      public BookOrder getNextOrder()
}
```



Step 3, implementation

```
// in this implementation, the oldest order is at the end of the array
public class OrdersCollection {
        private static final int MAX_ORDERS = 50000;
        private BookOrder[] orders = new BookOrder[MAX_ORDERS];
        private int numberOfOrders = 0;

        // adds an order to the collection
        public void addOrder(BookOrder order) {
                orders[numberOfOrders++] = order;
        }

        // returns and removes the oldest order
        public BookOrder getNextOrder() {
        // see next slide..
```


Step 3, implementation (cont'd)

```
// returns and removes the oldest order
public BookOrder getNextOrder() {
        BookOrder order = orders[0];

        // shift all orders one cell to the left
        System.arraycopy(orders, 1,
                         orders, 0,
                         numberOfOrders);

        numberOfOrders--;
        return order;
}
```

Our list element

- Remember:
  - The list holds a reference only to the head.
  - Each element references the next one.
- This means that our data type is not good enough (no 'next' member)
- We have two options:

1. Add a 'next' field to our existing class (convert BookOrder to an element)
2. "Wrap" our existing class with an 'element' class.

Option 1: converting to a list element

```
public class BookOrder {
     private Book book = null;
     private Customer customer = null;
     private BookOrder next = null;
     // constructors, getters and setters
}
```

Option 2: wrapping with an element

```
public class BookOrder {
	private Book book = null;
	private Customer customer = null;
	// constructors, getters and setters
}

public class BookOrderElement {
	private BookOrder data;
	private BookOrderElement next;
	// constructors, getters and setters
}
```


Which one is preferred?

- If we don't have access to the data type (if someone else is responsible for it), we have to use option #2.
- In order to use option #1, we have to redesign our class as a list element:
  - Name it BookOrderElement
  - Not expose it to customers, they don't care about elements.
- Option #2 is usually clearer and conforming with OOD.

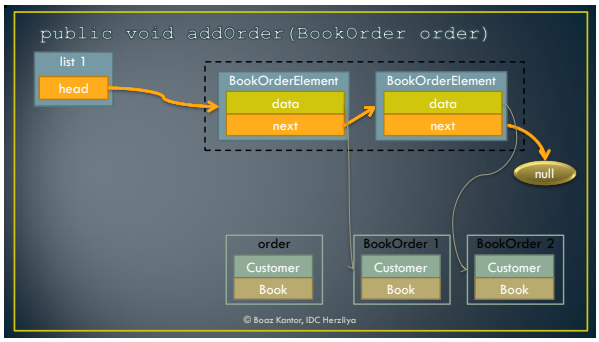
Linked list implementation (using option #2)

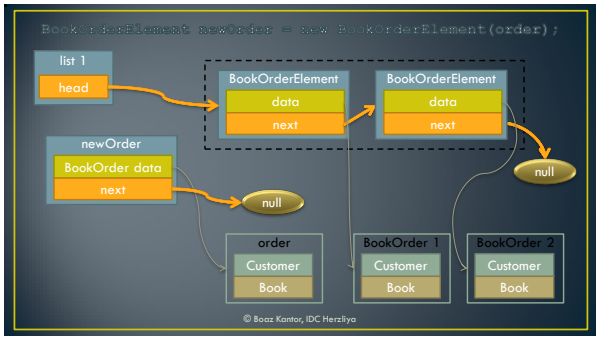
```
// in this implementation, the oldest order is at the beginning of the array
public class OrdersCollection {
	BookOrderElement head = null;

	// adds an order to the collection
	public void addOrder(BookOrder order) {
		BookOrderElement newOrder = new BookOrderElement(order);
		newOrder.setNext(this.head);
		head = newOrder;
	}

	// returns and removes the oldest order
	public BookOrder getNextOrder() {
	// see next slide..
}
```

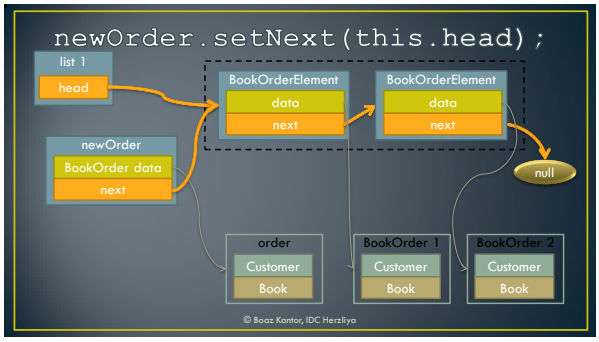
Linked list implementation (using option #2)

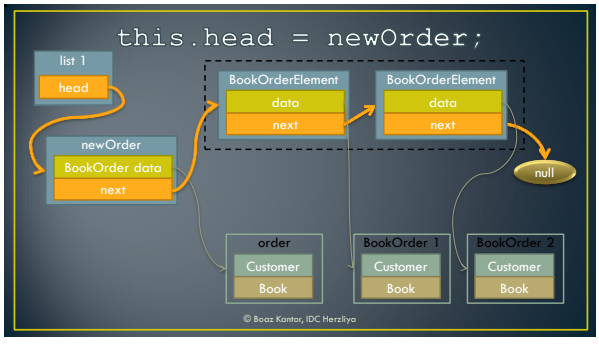
```
public BookOrder getNextOrder() {
        // TODO handle an empty list

        BookOrderElement previousElement = head;
        BookOrderElement currentElement = head;

        // TODO handle a special case where there is only one order in the list

        while (currentElement.getNext() != null) {
                previousElement = currentElement;
                currentElement = currentElement.getNext();
        }

        // remove the element from the list and return its data
        previousElement.setNext(null);
        return currentElement.getData();
}
```


LinkedList.addElement depiction




BookOrderElement newOrder = new BookOrderElement(order);
list 1
head
BookOrderElement
data
next
BookOrderElement
data
next
newOrder
BookOrder data
next
null
null
order
Customer
Book
BookOrder 1
Customer
Book
BookOrder 2
Customer
Book
© Boaz Kantor, IDC Herzliya


newOrder.setNext(this.head);
list 1
head
BookOrderElement
data
next
BookOrderElement
data
next
newOrder
BookOrder data
next
null
order
Customer
Book
BookOrder 1
Customer
Book
BookOrder 2
Customer
Book
© Boaz Kantor, IDC Herzliya


this.head = newOrder;
list 1
head
BookOrderElement
data
next
BookOrderElement
data
next
newOrder
BookOrder data
next
null
order
Customer
Book
BookOrder 1
Customer
Book
BookOrder 2
Customer
Book
© Boaz Kantor, IDC Herzliya

Queue

Concept

- FIFO: First In First Out
- Like in real life: "first come first served"
- The abstraction:
  - boolean isEmpty()
  - void enqueue(Element element)
  - Element dequeue(); // sometimes split in two:
    - void dequeue()
    - Element peek()

Implementation

- We can either use an array or our own linked list.
- If the queue size is finite/small/known in advance, we'll use an array. Otherwise, we'll be using a linked list.
- Upcoming: both implementations

Queue implementation #1, Array

```
public class Queue {
        private Element[] elements = new Element[MAX];
        private int numberOfElements = 0;
        public void enqueue(Element element) {
                elements[numberOfElements++] = element;
        }
        public Element dequeue() {
                Element result = elements[0];
                // shift all elements to the left…
                numberOfElements--;
                return result;
        }
        public boolean isEmpty() {
                return numberOfElements == 0;
        }
}
```


Queue implementation #2, Linked List

```
public class Queue {
        private LinkedList list = new LinkedList();
        public void enqueue(Element element) {
                list.insertAtBeginning(element);
        }
        public Element dequeue() {
                Element result = list.getFirst();
                list.remove(result);
                return result;
        }
        public boolean isEmpty() {
                return list.isEmpty();
        }
}
```


Summary (or: this is a total mess! Help!!)

- Arrays are inherent in the language
- Linked lists are based on custom classes
- Any other data structure can be implemented using:
  - Arrays
  - Lists
  - Other data structures
- Always find the most suitable data structure to implement your new data structure
- Always find the most suitable data structure to handle your data

Hands on thinking task

- Recall the book ordering system we wrote for Barnes & Noble:
  - We used array/linked list directly
  - A more suitable data structure would be the queue!
- Rewrite the Barnes & Noble ordering system using a queue

Boaz Kantor
Introduction to Computer Science,
Fall semester 2009-2010
IDC Herzliya

Introduction to Data Structures

Questions?