Abstract Data Types - Lists
Arrays implementation
Linked-lists implementation

Lecture 16
Jérôme Waldispühl
School of Computer Science
McGill University

Slides by Mathieu Blanchette
Recap from last lecture

• ADT: Model of a data structure that specifies:
  – The type of data stored
  – The operations supported on that data

• Application to lists (sequence of objects)

• We reviewed 2 types of implementations
  – Array-based implementation
  – Single-linked lists
Operations on list ADT

- **getFirst()**: returns the first object of the list
- **getLast()**: returns the last object of the list
- **getNth(n)**: returns the n-th object of the list
- **insertFirst(Object o)**: adds o at the beginning of the list
- **insertLast(Object o)**: adds o at the end of the list
- **insertNth(n, o)**: adds the n-th object of the list by o
- **removeFirst()**: removes the first object of the list
- **removeLast()**: removes the last object of the list
- **removeNth(n)**: removes the n-th object of the list
- **getSize()**: returns the number of objects in the list
- **concatenate(List l)**: appends List l to the end of this list
Array-based list ADT

- An 1D array \( L \) to store the elements of the list
- An integer \( \text{size} \) to record the number of objects stored.

<table>
<thead>
<tr>
<th>L:</th>
<th>O</th>
<th>G</th>
<th>C</th>
<th>N</th>
<th>Ø</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

\( \text{size} = 4 \)

- Easy to implement
- Space efficient
- the number of objects to be stored in not known in advance
- the user will need to do a lot of insertions or removals
Linked-list implementation

- Linked-list: Sequence of nodes. Each node stores some data and knows the next node in the list.
- A linked-list is a recursive data structure!

- Node:
  
- List:
/ Add an object at the tail of the list /*
void addLast(Object x) {
    if ( tail == null ) { // list is empty
tail = head = new node(x, head);
    }
else {
tail.setNext( new node(x,null) );
tail = tail.getNext();
}
}

Example: addLast( "Go!" )

```
"Go" | "Habs" | null

head  

"Go" | "Habs" | "Go!" |

head  tail
```
/* Add an object at the head of the list */
void addFirst(Object x) {
    head = new node(x, head);
    if (tail == null) tail = head;
}

Example: addFirst( "Go" )
insertNth(n, Object x) is more complicated...

Why? How to code it?

We will come back on that a bit later...

Example: insertNth(1, "Habs")
Example of utilization of linkedList

```java
public class testLists {

    public static void main(String args[]) {

        LinkedList l = new LinkedList(); // the list is empty for now
        l.addFirst("Roses");
        l.addLast("are");
        l.addLast("red");

        System.out.println(l.getFirst()); // prints Roses
        System.out.println(l.getLast()); // prints red

        ...
    }
}
```
Example: removeFirst()
class linkedList {
    node head, tail;

    ... // see previously defined methods

    removeFirst() { // You do it!
        if (head==null) return false; // the list was already empty
        head = head.getNext();
    }

    removeLast() { // You do it!
    }
Example: `removeLast()`
class linkedList {
    node head, tail;
    ...
    // see previously defined methods
    removeFirst() {
        // You do it!
        if (head==null) return false;  // the list was already empty
        head = head.getNext();
    }
    
    removeLast() {
        // You do it!
        if (head==null) return false;  // the list was already empty
        node newtail = head;
        while (newtail.getNext()! = tail) { newtail = newtail.getNext(); }  
        newtail.setNext(null);
        tail = newtail;
    }
}
Example: getNth(2)

N=0

"Go" | "Go!" | "Go!" | "Go!" | null

head

N=1

"Go" | "Go!" | "Go!" | "Go!" | null

head

tail

N=2

"Go" | "Go!" | "Go!" | "Go!" | null

head

tail
>Returns the first element of the list */

Object getFirst() {
    if (head==null) throw new Exception("getFirst: List empty!");
    return head.getValue();
}

>Returns the n-th elements of the list */

/* Runs in time O(n) */

Object getNth(int n) throws IndexOutOfBoundsException {
    if (n>=size()) throw new IndexOutOfBoundsException("n is too big!");
    node current=head;
    while (n>0) {
        current = current.getNext();
        n--;
    }
    return current;
}
Example: `insertNth( 1,"Habs" )`
/* insert Object at the n-th position of the list */
/* Runs in time O(n) */

boolean insertNth(int n, Object x) throws IndexOutOfBoundsException {
    if (n>=size()) throw new IndexOutOfBoundsException("n too big!");
    node predecessor = head;
    while (n>1) {
        predecessor = predecessor.getNext();
        n--;
    }

    node newelem = new node(x, predecessor.getNext());

    predecessor.setNext( newelem );

    return true;
}
Examples of utilization

```java
public class testLists {
    public static void main(String args[]) throws Exception {
        /* after the code listed before */
        System.out.println("The size is " + l.size());
        /* Since the get* methods return an Object, it needs to be cast to the correct type */
        String s = (String) l.getFirst();
        System.out.println("The zero-th element is " + s);
        System.out.println("The second element is " + l.getNth(1));
    }
}
```

The size is 3
The zero-th element is Rose
The second element is are

"Rose" | "are" | "red" | null

head

null
tail
Example: remove( "Habs" )
/* Removes from the list the first occurrence of object x. Returns true if x was removed. */

boolean remove(Object x) throws NoSuchElementException {
    if (head==null) throw new NoSuchElementException("List is empty!");
    if (head.getValue().equals(x)) {
        head=head.getNext();
        if (head==null) tail=null;
        return true;
    }

    node current = head;
    while (current.getNext()!=null && !current.getNext().getValue().equals(x))
        current = current.getNext();

    if (current.getNext()==null) return false;
    else {
        current.setNext(current.getNext().getNext());
        if (current.getNext()==null) tail=current;
    }
    return true;
}
Next Lectures

• **Stack ADT**: list that allows only operations at one end of the list
  – `push(object)`: inserts an element at the top of the stack
  – `object pop()`: removes the object at the top of the stack
  – `object top()`: returns the last inserted element
  – `integer size()`: returns the number of elements stored
  – `boolean isEmpty()`: indicates if stack is empty

• **Queues ADT**: List where insertion & removal are done on
  – `enqueue(object)`: inserts an element at the end of the queue
  – `object dequeue()`: removes the object at the front of the queue
  – `object front()`: returns the element at the front
  – `integer size()`: returns the number of elements stored
  – `boolean isEmpty()`: indicates if queue is empty