Abstract data types (ADT)

- Definition: Model of a data structure that specifies:
  - The type of data stored
  - The operations supported on that data

- An ADT specifies what can be done with the data, but not how it is done

- It is the implementation of the ADT that specifies how operations are performed

- The user of an ADT does not need to know anything about the implementation.

List ADT

Data stored: a ordered set of objects of any kind

\[[1, 1, 2, 3, 5, 8]\]
\[[\ln(), \sin(), f(), \exp()\]]
\[[\text{apple, pineapple, apple, banana}\]]

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): replaces 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

Implementation of the list ADT

With an Array

- An 1D array L to store the elements of the list
- An integer size to record the number of objects stored.

```
L = [apple, pineapple, apple, banana]
size = 4
```

```java
getFirst() { return L[0] }
getLast() { return L[size-1] }
getNth(n) { return L[n] }
```
Implementation of the list ADT

With an Array

insertLast( ) [  ,  ,  ]

Size = 3

insertLast( ) [  ,  ,  ,  ]

Size = 4

insertLast(Object o) { L[size] ← o; size ← size +1 }

insertNth(1, ) [  ,  ,  ,  ]

Size = 3

insertNth(1, ) [  ,  ,  ,  ]

Size = 4

insertNth(into n, Object o) {
for i ← size downto n {
L[i] ← L[i+1];
L[n] ← o;
size ← size +1 }
}

insertFirst(Object o) {
insertNth(0, o)
}

removeLast(): size ← size -1

removeNth(n)
for i ← n to size-1 do
L[i] ← L[i+1]
size ← size +1

removeFirst(): removeNth(0)

Limitations of arrays

- In some situations, an array is not a good way to implement a list, because:
  - Size has to be known in advance
  - Memory required may be much larger than the number of elements actually used
  - Inserting or deleting an element can take time up to O(n)

- An array implementation is bad when:
  - 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:

```
<table>
<thead>
<tr>
<th>Value</th>
<th>Next</th>
</tr>
</thead>
</table>
```

List:

```
head: | Go | Habs | Go |
null |
tail
```

public class node {
    private Object value;
    private node next;
    // constructor
    public node(Object x, node n) {
        value = x;
        next = n;
    }
    public node getNext() {               return next;          }
    public Object getValue() {           return value;        }
    public void setValue(Object x) {  value = x;            }
    public void setNext(node n) {      next = n;             }
}

public class node {
    private Object value;
    private node next;
    // constructor
    public node(Object x, node n) {
        value = x;
        next = n;
    }
    public node getNext() {               return next;          }
    public Object getValue() {           return value;        }
    public void setValue(Object x) {  value = x;            }
    public void setNext(node n) {      next = n;             }
}
```java
class linkedList {
    node head, tail;
    // default constructor, builds an empty list
    list() {
        head = null;
        tail = null;
    }
    getFirst() { return head.getValue(); }  
    getLast() { return tail.getValue(); }  
    getNth() { /* we will do later */ }
    ...

    // 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();
        }
    }

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

    insertNth(n, Object x) is more complicated…
    Why? How to code it?
    We will come back on that a bit later…

    Example: addLast( "Go!" )
    Example: addFirst( "Go" )
    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
        ...

        removeFirst(); // You do it!
        removeLast(); // You do it!
    }
}
```
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;
    }
    /*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: removeFirst()
    Example: removeLast()
    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 {
    node predecessor = head;
    while (n>1) {
        predecessor = predecessor.getNext();
        n--;
    }
    node newelem = new node(x, predecessor.getNext());
    predecessor.setNext( newelem );
    return true;
}

Visit the (n-1)-th first nodes in order to find the predecessor of n-th

Examples of utilization

public class testLists {
    public static void main(String args[]) throws Exception {
        /* after the code listed before */
        System.out.println("The size is "+l.size());
        String s = (String) l.getFirst();
        System.out.println("The zero-th element is "+s);
        System.out.println("The second element is "+l.getNth(1));
    }
}

/* Removes from the list the first occurrence of object x. Returns true if x was
   removed. */
boolean remove(Object x) throws NoSuchElementException {
    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;
}