LIST ADT

```
public interface List <T> {
  public boolean isEmpty() ;
  //effects: Restituisce true se la lista e'vuota altrimenti
     false
  public int size();
//Restituisce I numero degli elementi nella lista
  public T get (int givenPos)
            throws ListIndexOutOfBoundException
//GivenPos e' una posizione nella lista degli elementi
//Se 1<= givenPos<= size() allora l'elemento a givenPos
     e' restituito
// Throws??
```

```
public interface List <T> {
  public boolean isEmpty() ;
  //effects: Restituisce true se la lista e'vuota altrimenti
     false
  public int size();
//Restituisce I numero degli elementi nella lista
  public T get (int givenPos)
            throws ListIndexOutOfBoundException
//GivenPos e' una posizione nella lista degli elementi
//Se 1<= givenPos<= size() allora l'elemento a givenPos
     e' restituito
// Throws: givenPos<1 o givenPos >= size()+1
```

```
Public void add(int givePsos, T newItem)
     throws ListOutOfBoundException, List Exception
//Requires: GivenPos indica la posizione dove deve
     essere inserito il nuovo elemento
//Effect: se 1 <=  givenPos <= size() +1, newItem e' alla
     posizione givenPos, gli element di posizione
     maggiori di givenPos sono spostati di una posizione
//Throws: LOBE givenPos < 1 or givenPos > size()
Throws: LE: newItem non puo' essere inserito
```

```
public void remove (int givenPos)
   throws ListOutOfBoundException;
//Requires: givenPos indica la posizione
    dell'elemento che deve essere rimosso
//Effect: se 1 <=  givenPos <= size() , l'elemento  alla
    posizione givenPos e' rimosso , gli element di
    posizione maggiori di givenPos sono spostati di
    una posizione a sinistra
//Throws: givenPos < 1 or givenPos > size()
```

```
public class ListOutOfBoundException
  extend OutOfBoundException{
    public ListOutOfBoundException (string s) {
        super(s)
    }
}
```

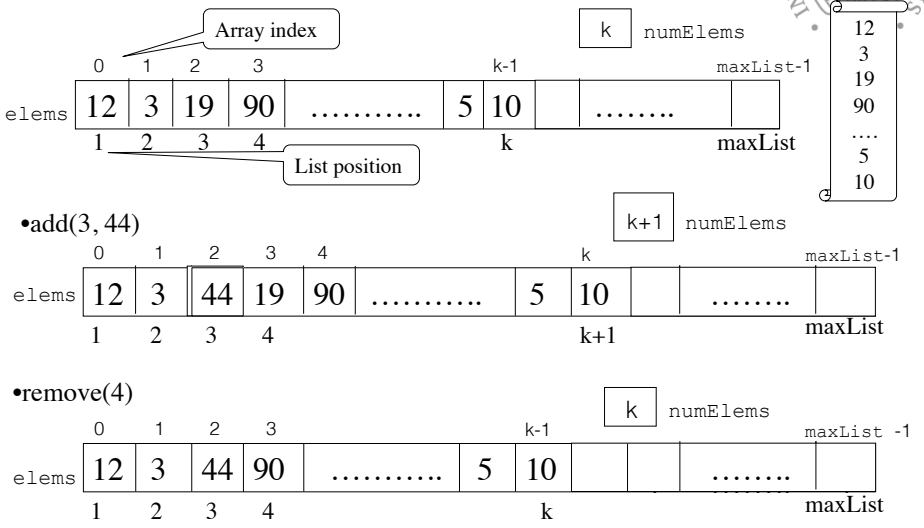
```
public class ListException
  extend RunTimeException{
    public ListException (string s) {
        super(s)
    }
}
```

Static Implementation

- Array di dimensione fissata a priori
- L'indice dell'array indica la posizione nella lista
- Strutture di implementazione

```
private final int maxList = 100;
private T elems[maxList];
private int numElem;
```

Data structure and operations



Array-based Implementation of List

```
public class ArrayBasedList<T> implements List<T>{

    private static final int maxList = 50;
    private T[] elems;                          // a list of elements
    private int numElems;                       // current number of elements in the list

    public ArrayBasedList(){
     elems= (T[]) new Object[maxList];
     numElems = 0;
    }
    public boolean isEmpty(){
     return (numElems == 0);}

    public int size(){
        return numElems;}


    public T get(int givenPos) throws
                              ListIndexOutOfBoundsException{
     if (givenPos >= 1 && givenPos <= numElems) {
            return elems[translate(givenPos)];
        }
     else {throw new ListIndexOutOfBoundsException("Position out of range");}
    }// end get
                                                                          ....
```

Array-based Implementation of List

```
public void add(int givenPos, T newItem) throws
                        ListIndexOutOfBoundsException, ListException {
    if (numElems == maxList) {
          throw new ListException("List is full"); }
    if (givenPos >=1 && givenPos <= numElems + 1){
        makeRoom(givenPos);
        elems[translate(givenPos)]= newItem;        //insert newItem
        numElems++;
    }
    else throw new ListIndexOutOfBoundsException("Position out of range");
}// end add


private void makeRoom(int position) {
//pre: 1 <= position <= numElems + 1
    for (int pos=numElems; pos>=position; pos--) {
      elems[translate(pos+1)] = elems[translate(pos)]; }
}

private int translate(int position){
     return position -1;
}//end translate
```

Continued ….

Array-based Implementation of List

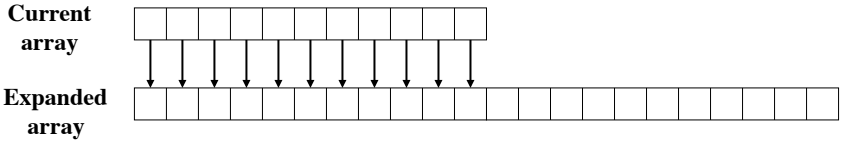
```
public void remove(int givenPos) throws ListIndexOutOfBoundsException{
    if (givenPos >=1 && givenPos <= numElems){
       if (givenPos < numElems){
        // delete item by shifting left all item at position > givenPos
        removeGap(givenPos);
       }
       numElems--;
    }
    else throw new ListIndexOutOfBoundsException("Position out of range");
}// end remove


private void removeGap(int position){
//pre: 1=< position < numElems
    for (int pos=position+1; pos<=size(); pos++){
       elems[translate(pos-1)] = elems[translate(pos)];}
}

}//end ListArrayBased
```

Alternativa

Array dinamici



- Operazione di copia: costo computazionale

con controllo della memoria.

Lists:Java Collection

Java fornisce una interfaccia simila a quella che abbiamo visto `java.util.List`

```
public interface List<E> extends Collection<E>
```

Java fornisce diverse implementazioni.

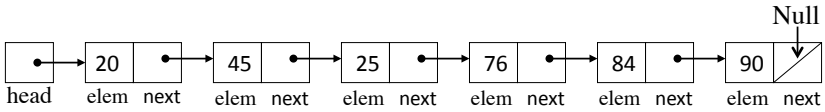
`java.util.ArrayList<E>` represents each list by an array.

`java.util.LinkedList<E>` represents each list by a doubly-linked list.

Queste implementazioni non sono synchronized!!.

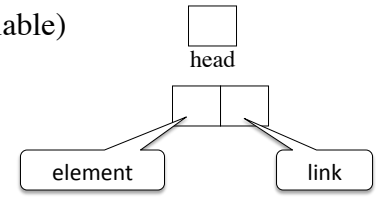
List ADT: Dynamic Implementation

Linked list:



Strutture dati:

- una variable (reference variable) di tipo node
- node data type



class Node

Package level class

```
public class Node<T>{
   private  T element;
   private Node<T> next;

   public void setElem(T newElem){
     element = newElem;
   }
   public T getElem( ){
     return element;
   }
   public void setNext(Node<T>
             newNode){
     next = newNode;
   }
   public Node<T> getNext( ){
     return next;
   }
}
```

Inner class

```
private class Node{
   private  T element;
   private Node next;

   private Node(T newElem){
     element = newElem;
     next = null;
   }

   private Node(T newElem,
                Node newNode){
     element = newElem;
     next = newNode;
   }
}
```

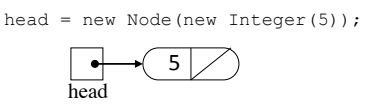
LinkedBasedList(1)

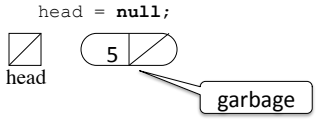
```
public class LinkedBasedList<T> implements List<T>{
   private Node head;
   private int numElems;
   public LinkedBasedList(){
    head = null;
    numElems = 0;
   }
   <<Implementation of public isEmpty, size,
    get, add, remove go here>>
   private Node getNodeAt(int givenPos)
   <implementation deferred>
   private class Node{
    << >>
   }
}
```

LinkedBasedList(2)

Invariante

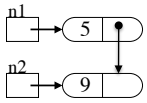
- Campo `next` dell'ultimo nodo deve essere `null`.
- quando la lsita e' vuota la variabile `head` deve essere `null`.



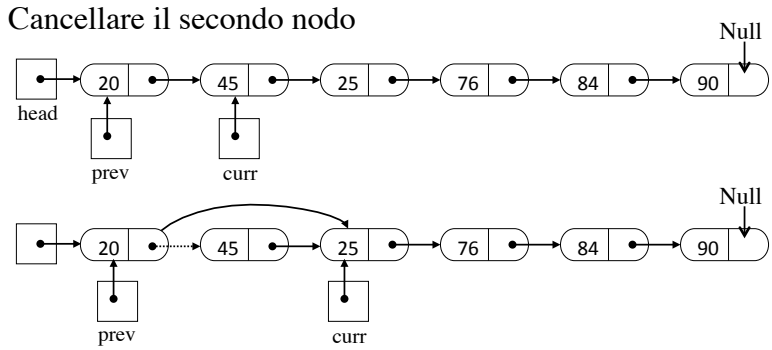


Codice

```
Node<Integer> n1 = new Node( );
Node<Integer> n2 = new Node( );
n1.setElem(new Integer(5));
n2.setElem(new Integer(9));
n1.setNext(n2);
```



Esempio: rimozione



LinkedBasedList (2)

```
public class LinkedBasedList<T> implements List<T>{
   private Node<T> head;
   private int numElems;
   <constructors and methods go here>
}
```

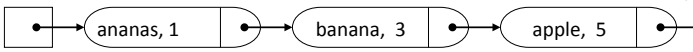
- Variables "prev" e"curr" sono locali ai metodi
  non sono variabili di istanza di LinkedBasedList<T>.
- method "`getNodeAt(i)`" ausiliario restituisce il puntatore all'i-esimo elemento della lista.

```
public LinkedBasedList( ){
    numElems = 0;
    head = null;
}
```

```
public boolean isEmpty( ){
    return numElems == 0;
}
```

```
private Node<T> getNodeAt(int givenPos){
   // requires: givenPos is the position of the desired node,
   // requires: assume 1 ≤ givenPos ≤ numElems;
   // efffects: returns reference to the node at a givenPos;

   Node<T> curr = head;
   for (int skip = 1; skip < givenPos; skip++){
           curr = curr.getNext();
   }
   return curr;
}
```




```
public class Entry {
    private  String element;
    private int amount;

    public Entry(String elementData,
                    int amountData){
     element = elementData;
     amount = amountData;
    }

    public String toString( ){
     return (element + " " + amount);
    }
```

```
List has 3 elements.
ananas1
banana 3
apples 5
End of list.
```

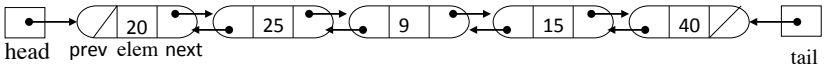
```
public class GenericLinkedListDemo {

  public static void main(String[]
                                  args){

  List<Entry> list =
     new LinkedBasedList<Entry>( );
  Entry e1 = new Entry("apple", 5);
  Entry e2 = new Entry("banana", 5);
  Entry e3 = new Entr("ananas",1);
  list.add(1, e1);
  list.add(1, e2);
  list.add(1, e3);
  System.out.println(" List has " +
         list.size() + " elements. ");
  list.diplay();
  System.out.println("End of list. ");
  }
}
```

Variazione

Doubly – Linked Lists



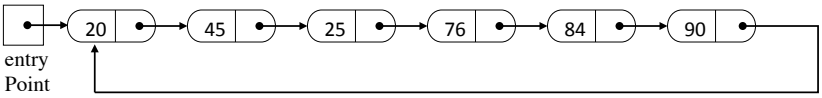
Node: previous node e next node

Node<T>

```
public class Node<T>{
    private  T item;
    private  Node<T> next;
    private  Node<T> previous;
        < constructors, accessor and
        mutator methods here>
}
```

Variazioni

Circular Lists



Ordered Linked List

Ogni elemento e'associato con un particolare attributo detto chiave, key, gli elementi sono totalmente ordinati rispetto alla chiave

createOrderedList( )
// effecr: Create an empty ordered list

put(searchKey, givenValue)
//Insert in the list an element with searchKey
//and givenValue in the correct position.
//If a node with searchKey exists, set its
//value to be equal to givenValue.

remove(searchKey)
//effect: Removes the element whose key is equal
//to searchKey. Throws exception if such an
//element does not exist.

isEmpty( )
// effect: Determine if an ordered list is
//empty

get(searchKey)
//effect: Returns the element with key equal
//to searchKey. Returns null if the searchKey
//is not in the list

size( )
// effect: Returns the number of elements in
// an ordered list

Ricerca elemento

codice

```
get(K searchKey){
// post: search the Node whose key is equal to searchKey
// and returns its value.
Let prev be equal to head;
if (prev is null)
  { the desired Node is not found, return null; }
else if (prev.key is equal to searchKey)
      { the desired Node is found, return its value; }
      else {curr is equal to the next node;
            while (curr is different from null && curr.key is less then searchKey){
                  pre is equal to curr;
                   curr is equal to the next node;
             }
            if (curr is different from null and curr.key is equal to searchKey){
                  return curr.value;}
            }
      return null;
}
```

You can use here a method **equals(searchKey)**

You can use here a method **compareTo(searchKey)**

Lists implementazioni statiche o dinamiche :

Operations	Fixed-size Array	Linked List
add(pos, elem)	O(n) to O(1)	O(1) to O(n)
remove(pos)	O(n) to O(1)	O(1) to O(n)
get(pos)	O(1)	O(1) to O(n)
display()	O(n)	O(n)
size(), isEmpty()	O(1)	O(1)