Lists and Iterators

The java.util.List ADT

- The java.util.List interface includes the following methods:

size( ): Returns the number of elements in the list.

isEmpty( ): Returns a boolean indicating whether the list is empty.

get(i): Returns the element of the list having index i; an error condition occurs if i is not in range $[0, \text{size}( ) - 1]$.

set(i, e): Replaces the element at index i with e, and returns the old element that was replaced; an error condition occurs if i is not in range $[0, \text{size}( ) - 1]$.

add(i, e): Inserts a new element e into the list so that it has index i, moving all subsequent elements one index later in the list; an error condition occurs if i is not in range $[0, \text{size}( )]$.

remove(i): Removes and returns the element at index i, moving all subsequent elements one index earlier in the list; an error condition occurs if i is not in range $[0, \text{size}( ) - 1]$.

Example

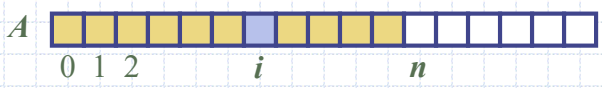
- A sequence of List operations:

Method	Return Value	List Contents
add(0, A)	–	(A)
add(0, B)	–	(B, A)
get(1)	A	(B, A)
set(2, C)	"error"	(B, A)
add(2, C)	–	(B, A, C)
add(4, D)	"error"	(B, A, C)
remove(1)	A	(B, C)
add(1, D)	–	(B, D, C)
add(1, E)	–	(B, E, D, C)
get(4)	"error"	(B, E, D, C)
add(4, F)	–	(B, E, D, C, F)
set(2, G)	D	(B, E, G, C, F)
get(2)	G	(B, E, G, C, F)

3

Array Lists

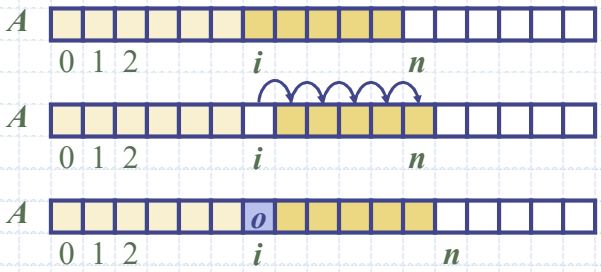
- An obvious choice for implementing the list ADT is to use an array, **A**, where **A[i]** stores (a reference to) the element with index **i**.
- With a representation based on an array **A**, the get(**i**) and set(**i**, **e**) methods are easy to implement by accessing **A[i]** (assuming **i** is a legitimate index).



4

Insertion

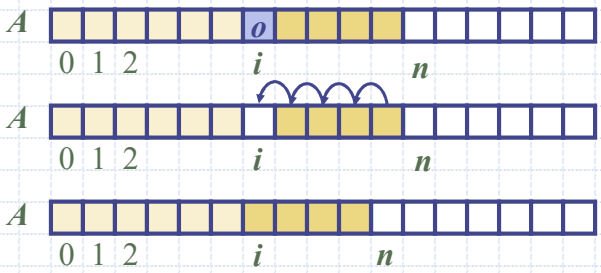
- In an operation *add*(i, o), we need to make room for the new element by shifting forward the $n - i$ elements $A[i], \ldots, A[n-1]$
- In the worst case ($i = 0$), this takes $O(n)$ time



5

Element Removal

- In an operation *remove*(i), we need to fill the hole left by the removed element by shifting backward the $n - i - 1$ elements $A[i+1], \ldots, A[n-1]$
- In the worst case ($i = 0$), this takes $O(n)$ time



6

Performance

- In an array-based implementation of a dynamic list:
  - The space used by the data structure is $O(n)$
  - Indexing the element at i takes $O(1)$ time
  - *add* and *remove* run in $O(n)$ time
- In an *add* operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one …

Java Implementation

```
11    // public methods
12    /** Returns the number of elements in the array list. */
13    public int size( ) { return size; }
14    /** Returns whether the array list is empty. */
15    public boolean isEmpty( ) { return size == 0; }
16    /** Returns (but does not remove) the element at index i. */
17    public E get(int i) throws IndexOutOfBoundsException {
18      checkIndex(i, size);
19      return data[i];
20    }
21    /** Replaces the element at index i with e, and returns the replaced element. */
22    public E set(int i, E e) throws IndexOutOfBoundsException {
23      checkIndex(i, size);
24      E temp = data[i];
25      data[i] = e;
26      return temp;
27    }
```

Java Implementation, 2

```
28    /** Inserts element e to be at index i, shifting all subsequent elements later. */
29    public void add(int i, E e) throws IndexOutOfBoundsException,
30                                       IllegalStateException {
31      checkIndex(i, size + 1);
32      if (size == data.length)                  // not enough capacity
33        throw new IllegalStateException("Array is full");
34      for (int k=size-1; k >= i; k--)           // start by shifting rightmost
35        data[k+1] = data[k];
36      data[i] = e;                              // ready to place the new element
37      size++;
38    }
39    /** Removes/returns the element at index i, shifting subsequent elements earlier. */
40    public E remove(int i) throws IndexOutOfBoundsException {
41      checkIndex(i, size);
42      E temp = data[i];
43      for (int k=i; k < size-1; k++)            // shift elements to fill hole
44        data[k] = data[k+1];
45      data[size-1] = null;                      // help garbage collection
46      size--;
47      return temp;
48    }
49    // utility method
50    /** Checks whether the given index is in the range [0, n-1]. */
51    protected void checkIndex(int i, int n) throws IndexOutOfBoundsException {
52      if (i < 0 || i >= n)
53        throw new IndexOutOfBoundsException("Illegal index: " + i);
54    }
55  }
```

Growable Array-based Array List

- Let push(o) be the operation that adds element o at the end of the list
- When the array is full, we replace the array with a larger one
- How large should the new array be?
  - Incremental strategy: increase the size by a constant c
  - Doubling strategy: double the size

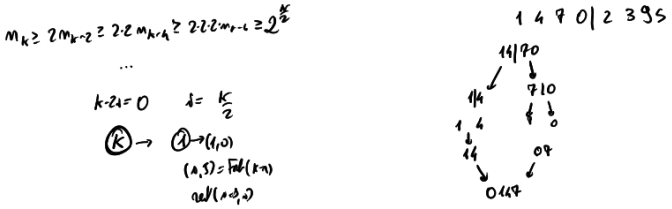
Algorithm *push*(o)
  if $t = S.length - 1$ **then**
    $A \leftarrow$ **new array of size …**
    for $i \leftarrow 0$ **to** $n-1$ **do**
      $A[i] \leftarrow S[i]$
    $S \leftarrow A$
  $n \leftarrow n + 1$
  $S[n-1] \leftarrow o$

Growable Array-based Array List (Java)

```
public void add (int i, E e) throws IndexOutOfBoundsException {
   checkIndex(i, size + 1);
   if (size == data.length)              // not enough capacity
     resize(2 * data.length);            // so double the current capacity
   for (int k=size-1; k >= i; k--)       // start by shifting rightmost
     data[k+1] = data[k];
   data[i] = e;                          // ready to place the new element
   size++;
  }
```

```
protected void resize (int capacity) {
   E[] temp = (E[]) new Object[capacity];
for (int k=0; k < size; k++)
     temp[k] = data[k];
   data = temp;                          // start using the new array
    }
  }
```

11



Positional Lists

- To provide for a general abstraction of a sequence of elements with the ability to identify the location of an element, we define a **positional list** ADT.
- A position acts as a marker or token within the broader positional list.
- A position p is unaffected by changes elsewhere in a list; the only way in which a position becomes invalid is if an explicit command is issued to delete it.
- A position instance is a simple object, supporting only the following method:
  - P.getElement( ): Return the element stored at position p.

```
public interface Position<E> {
 /*Returns the element stored at this position.
  * @return the stored element
  * @throws IllegalStateException if position no longer valid*/
 E getElement() throws IllegalStateException;
}
```

12

Positional List ADT

Riferirsi alle posizioni

- Accessor methods:

first( ): Returns the position of the first element of L (or null if empty).

last( ): Returns the position of the last element of L (or null if empty).

before(p): Returns the position of L immediately before position p (or null if p is the first position).

after(p): Returns the position of L immediately after position p (or null if p is the last position).

isEmpty( ): Returns true if list L does not contain any elements.

size( ): Returns the number of elements in list L.

13

Riusciamo a seguire un oggetto che metto nella lista. Non come singly Linked list.

Positional List ADT, 2

- Update methods:

addFirst(e): Inserts a new element e at the front of the list, returning the position of the new element.

addLast(e): Inserts a new element e at the back of the list, returning the position of the new element.

addBefore(p, e): Inserts a new element e in the list, just before position p, returning the position of the new element.

addAfter(p, e): Inserts a new element e in the list, just after position p, returning the position of the new element.

set(p, e): Replaces the element at position p with element e, returning the element formerly at position p.

remove(p): Removes and returns the element at position p in the list, invalidating the position.

14

Mette questo elemento dopo quello puntato da p.

Crea un puntatore posizione: tag associato al valore 8.

Example

- A sequence of Positional List operations:

Method	Return Value	List Contents
addLast(8)	p	(8_p)
first( )	p	(8_p)
addAfter(p, 5)	q	$(8_p, 5_q)$
before(q)	p	$(8_p, 5_q)$
addBefore(q, 3)	r	$(8_p, 3_r, 5_q)$
r.getElement( )	3	$(8_p, 3_r, 5_q)$
after(p)	r	$(8_p, 3_r, 5_q)$
before(p)	null	$(8_p, 3_r, 5_q)$
addFirst(9)	s	$(9_s, 8_p, 3_r, 5_q)$
remove(last( ))	5	$(9_s, 8_p, 3_r)$
set(p, 7)	8	$(9_s, 7_p, 3_r)$
remove(q)	"error"	$(9_s, 7_p, 3_r)$

15

Java Implementation (Positional List Interface)

```
public interface PositionalList<E> extends Iterable<E> {

  int size();

  boolean isEmpty();

  Position<E> first();

  Position<E> last();

  Position<E> before(Position<E> p) throws IllegalArgumentException;

  Position<E> after(Position<E> p) throws IllegalArgumentException;

  Position<E> addFirst(E e);

  Position<E> addLast(E e);

  Position<E> addBefore(Position<E> p, E e)
    throws IllegalArgumentException;

  Position<E> addAfter(Position<E> p, E e)
    throws IllegalArgumentException;

  E set(Position<E> p, E e) throws IllegalArgumentException;

  E remove(Position<E> p) throws IllegalArgumentException;

  Iterator<E> iterator();

  Iterable<Position<E>> positions();
}
```

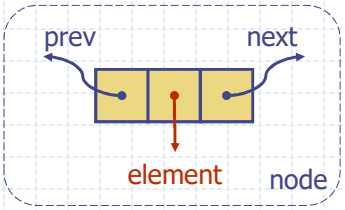
16

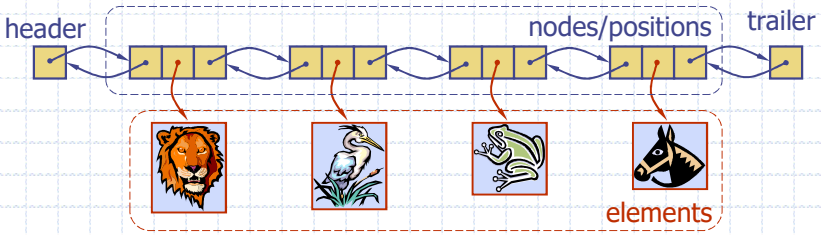
→ Scorro le posizioni in questo modo

Restituisce classe iterable che ci consente di attraversare lista attraverso le posizioni.

Positional List Implementation

- The most natural way to implement a positional list is with a doubly-linked list.

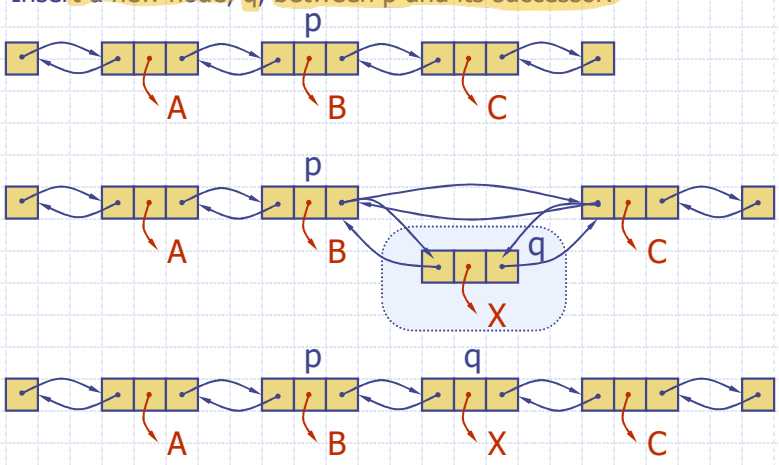




17

Insertion

- Insert a new node, q, between p and its successor.



18

Java Implementation (Positional List)

```
35
36  public class LinkedPositionalList<E> implements PositionalList<E> {
37
38    private static class Node<E> implements Position<E> {
39
40      private E element;               // reference to the element stored at this node
41      private Node<E> prev;            // reference to the previous node in the list
42      private Node<E> next;            // reference to the subsequent node in the list
43
44      public Node(E e, Node<E> p, Node<E> n) {
45        element = e;
46        prev = p;
47        next = n;}
48
49      public E getElement() throws IllegalStateException {
50       if (next == null)                 // convention for defunct node
51         throw new IllegalStateException("Position no longer valid");
52       return element;}
53
54      public Node<E> getPrev() {return prev;}
55      public Node<E> getNext() { return next;}
56      public void setElement(E e) {element = e;}
57      public void setPrev(Node<E> p) {prev = p; }
58      public void setNext(Node<E> n) {next = n;}
59    } //----------- end of nested Node class -----------
60
61    private Node<E> header;                      // header sentinel
62    private Node<E> trailer;                     // trailer sentinel
63    private int size = 0;                        // number of elements in the list
64
65    /** Constructs a new empty list. */
66    public LinkedPositionalList() {
67      header = new Node<>(null, null, null);     // create header
68      trailer = new Node<>(null, header, null);  // trailer is preceded by header
69      header.setNext(trailer);                   // header is followed by trailer
70    }
```

Non ci sono nodi che puntano a null

Controlla se è un nodo

Java Implementation (Positional List)

```
private Node<E> validate(Position<E> p) throws IllegalArgumentException {
  if (!(p instanceof Node)) throw new IllegalArgumentException("Invalid p");
  Node<E> node = (Node<E>) p;      // safe cast
  if (node.getNext() == null)      // convention for defunct node
    throw new IllegalArgumentException("p is no longer in the list");
  return node;
}
private Position<E> position(Node<E> node) {
  if (node == header || node == trailer)
    return null;   // do not expose user to the sentinels
  return node;
}
public int size() { return size; }

public boolean isEmpty() { return size == 0; }

public Position<E> first() {return position(header.getNext());}

public Position<E> last() {return position(trailer.getPrev()); }

private Position<E> addBetween(E e, Node<E> pred, Node<E> succ) {
  Node<E> newest = new Node<>(e, pred, succ);  // create and link a new node
  pred.setNext(newest);
  succ.setPrev(newest);
  size++;
  return newest;
}
public Position<E> addFirst(E e) {
  return addBetween(e, header, header.getNext());        // just after the header
}
public Position<E> addLast(E e) {
  return addBetween(e, trailer.getPrev(), trailer);      // just before the trailer
}
public Position<E> addBefore(Position<E> p, E e)
                              throws IllegalArgumentException {
  Node<E> node = validate(p);
  return addBetween(e, node.getPrev(), node);
}
public Position<E> addAfter(Position<E> p, E e)
                              throws IllegalArgumentException {
  Node<E> node = validate(p);
  return addBetween(e, node, node.getNext());
}
public E set(Position<E> p, E e) throws IllegalArgumentException {
  Node<E> node = validate(p);
  E answer = node.getElement();
  node.setElement(e);
  return answer;
}
```

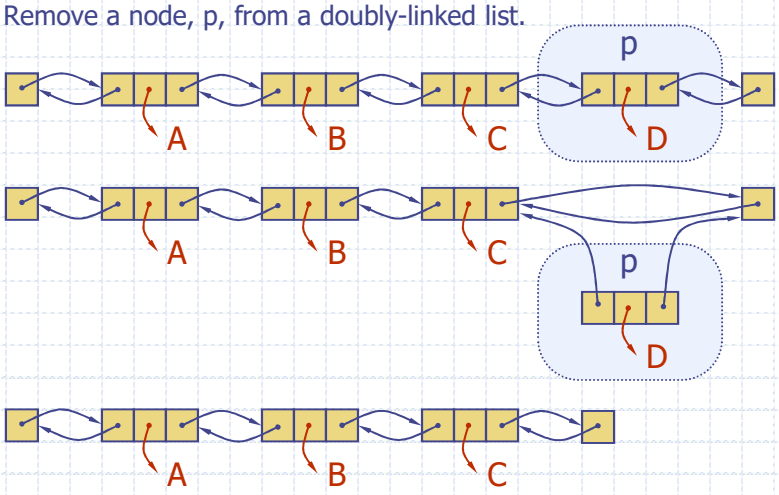
evita di dare

Metodi che hanno bisogno della posizione di riferimento: devo validare la posizione

La funzione position evita che ritorni in uscita header o trailer.

Deletion

- Remove a node, p, from a doubly-linked list.



21

Java Implementation (Positional List)

```
/**
 * Removes the element stored at the given Position and returns it.
 * The given position is invalidated as a result.
 *
 * @param p the Position of the element to be removed
 * @return the removed element
 * @throws IllegalArgumentException if p is not a valid position for this list
 */
@Override
public E remove(Position<E> p) throws IllegalArgumentException {
  Node<E> node = validate(p);
  Node<E> predecessor = node.getPrev();
  Node<E> successor = node.getNext();
  predecessor.setNext(successor);
  successor.setPrev(predecessor);
  size--;
  E answer = node.getElement();
  node.setElement(null);          // help with garbage collection
  node.setNext(null);             // and convention for defunct node
  node.setPrev(null);
  return answer;
}
```

Sempre validazione

22

Java Implementation (Positional List)

```
1  import java.io
2  import stdio.*;
3  import net.data
4
5  public class Te
6
7
8
9
10  public static
11 {
12
13
14  LinkedPositionalList listapos= new LinkedPositionalList<String>();
15
16  String[] testo = {"primo","secondo","terzo"};
17 System.out.println("Avvio lista");
18
19
20 for (String s: testo) {
21     listapos.addLast(s);
22 }
23 String s1="primonuovo";
24 listapos.set(listapos.first(),s1);
25
26
27 System.out.println("Dimensione lista= "+ listapos.size());
28 System.out.println("Primo lista= "+ listapos.first().getElement());
29 Position p = listapos.after(listapos.first());
30 System.out.println("Elemento successivo al primo= "+ p.getElement());
31 }
32 }
```

```
algoritmi miei — -bash — 82×12
Last login: Wed Mar 27 11:12:57 on console
iMac-di-Rocco:algoritmi miei roccoaversa$ java TestListPos
Avvio lista
Dimensione lista= 3
Primo lista= primonuovo
Elemento successivo al primo= secondo
iMac-di-Rocco:algoritmi miei roccoaversa$
```

Dobbiamo rendere lista una collection = classe che implementa Iterable

The for-each Loop

- Java's Iterable class also plays a fundamental role in support of the "for-each" loop syntax:

```
for (ElementType variable : collection) {
  loopBody                                  // may refer to "variable"
}
```

Iterable class

is equivalent to:

```
Iterator<ElementType> iter = collection.iterator();
while (iter.hasNext()) {
  ElementType variable = iter.next();
  loopBody                                  // may refer to "variable"
}
```

24

Restituisce un oggetto che implementa Iterator.

↓

ha metodi next, hasnext ecc.

next e hasnext => Devo scegliere un modo di attraversarlo

The Iterable Interface

- Java defines a parameterized interface, named Iterable, that includes the following single method:
  - iterator( ): Returns an *Iterator* of the elements in the collection.
- An instance of a typical collection class in Java, such as an ArrayList, is iterable (but not itself an iterator); it produces an Iterator for its collection as the return value of the iterator( ) method.
- Each call to iterator( ) returns a new Iterator instance, thereby allowing multiple (even simultaneous) traversals of a collection.

→ implementa Iterator

25

Iterators

- An Iterator is a software design pattern that abstracts the process of scanning through a sequence of elements, one element at a time.

hasNext( ): Returns true if there is at least one additional element in the sequence, and false otherwise.

next( ): Returns the next element in the sequence.

remove() Remove the current element of the sequence.

26

Iterator é único obviamente.

Support for iterating positions in a Positional List

- A (nonstatic) inner class PositionIterator.

Note well that each instance contains an implicit reference to the containing list allowing us to call the list's methods directly.

La attraverso per posizioni

implementa iterator!

Java Implementation (Position Iterator – nested class)

```
154  private class PositionIterator implements Iterator<Position<E>> {
155
156    /** A Position of the containing list, initialized to the first position. */
157    private Position<E> cursor = first();  // position of the next element to report
158    /** A Position of the most recent element reported (if any). */
159    private Position<E> recent = null;     // position of last reported element
160
161
162    public boolean hasNext() { return (cursor != null); }
163
164
165    public Position<E> next() throws NoSuchElementException {
166      if (cursor == null) throw new NoSuchElementException("nothing left");
167      recent = cursor;              // element at this position might later be removed
168      cursor = after(cursor);
169      return recent;
170    }
171
172  } //------------ end of nested PositionIterator class ------------
173
174  //---------------- nested PositionIterable class ----------------
175  private class PositionIterable implements Iterable<Position<E>> {
176    public Iterator<Position<E>> iterator() { return new PositionIterator(); }
177  } //------------ end of nested PositionIterable class ------------
178
179
180
181  public Iterable<Position<E>> positions() {
182    return new PositionIterable();       // create a new instance of the inner class
183  }
184
```

null if empty

Next restituisce valore puntato corrente: al primo utilizzo dà il primo valore. Quando mi fermo? Devo vedere cursor, che punta elemento corrente

remove { remove(cursor) }

→ classe che implementa Iterable → Restituisce PositionIterable

→ Possiamo attraversare per posizioni.

Per scorrere posizioni: NomeLista.positions() è la collezione da mettere nel for each.

Java Implementation (Use Position Iterator)

```
import java.io.*;
import java.util.*;
import stdio.*;
import net.datastruc

public class TestLis



 public static void
{


 LinkedPositionalLi
 // Iterator<String>
 String[] testo = {
System.out.println(


for (String s: teste
    listapos.addLast
}
String s1="primonuovo";
listapos.set(listapos.first(),s1);


System.out.println("Dimensione lista= "+ listapos.size());
System.out.println("Primo lista= "+ listapos.first().getElement());
Position p = listapos.after(listapos.first());
System.out.println("Elemento successivo al primo= "+ p.getElement());

 Iterable<Position<String>> iter2=listapos.positions();
 for (Position p1: iter2 ) {
    System.out.println("Valore nella lista per posizione = "+ p1.getElement());
 }

Iterator<Position<String>> iter3=iter2.iterator();
while  (iter3.hasNext()) {
    System.out.println("Valore nella lista= "+ iter3.next().getElement());
 }
}
}
```

```
algoritmi miei — -bash — 70×15
Last login: Wed Mar 27 12:07:17 on ttys000
iMac-di-Rocco:algoritmi miei roccoaversa$ java TestListPos
Avvio lista
Dimensione lista= 3
Primo lista= primonuovo
Elemento successivo al primo= secondo

Valore nella lista per posizione = primonuovo
Valore nella lista per posizione = secondo
Valore nella lista per posizione = terzo
Valore nella lista= primonuovo
Valore nella lista= secondo
Valore nella lista= terzo
iMac-di-Rocco:algoritmi miei roccoaversa$
```

29

Creo più collezioni e le scorro una dopo l'altra.

Support for iterating elements in a Positional List

- A (nonstatic) inner class ElementIterator.

This class adapts use the PositionIterator to iterate the elements of the list.

30

Java Implementation (Element Iterator nested class)

```
394
395
396  /* This class adapts the iteration produced by positions() to return elements. */
397  private class ElementIterator implements Iterator<E> {
398    Iterator<Position<E>> posIterator = new PositionIterator();
399    public boolean hasNext() { return posIterator.hasNext(); }
400    public E next() { return posIterator.next().getElement(); } // return element!
401    public void remove() { posIterator.remove(); }
402  }
403
404  /**
405   * Returns an iterator of the elements stored in the list.
406   * @return iterator of the list's elements
407   */
408  @Override
409  public Iterator<E> iterator() { return new ElementIterator(); }
410
411
412
```

31

Java Implementation (Use Element Iterator)

```
import java.io.*;
import java.util.*;
import stdio.*;
import net.datastructures.*;

public class TestListPos {



 public static void main(String argv[])
{


 LinkedPositionalList listapos= new LinkedPositionalList<String>();
 // Iterator<String> iter;
 String[] testo = {"primo","secondo","terzo"};
System.out.println("Avvio lista");


for (String s: testo) {
    listapos.addLast(s);
}
String s1="primonuovo";
listapos.set(listapos.first(),s1);


System.out.println("Dimensione lista= "+ listapos.size());
System.out.println("Primo lista= "+ listapos.first().getElement());
Position p = listapos.after(listapos.first());
System.out.println("Elemento successivo al primo= "+ p.getElement());

Iterator<String> iter=listapos.iterator();
while  (iter.hasNext()) {
    System.out.println("Valore nella lista= "+ iter.next());
 }

}
}
```

32

Java Implementation (Use Element Iterator)

```
import java.io.*;
import java.util.*;

import net.datastructures.*;

public class TestListPos_interi {


 public static void main(String argv[])
{


 LinkedPositionalList<Double> listapos= new LinkedPositionalList<Double>();

Double[] dati = {100.0,200.0,300.0};
System.out.println("Avvio lista");



for (Double i: dati) {
    listapos.addLast(i);
}

System.out.println("Dimensione lista= "+ listapos.size());
System.out.println("Primo lista= "+ listapos.first().getElement());
for (Double slp: listapos ) {
    System.out.println("Valore nella lista (collezione) = "+ slp);
 }
}
}
```

33

Alternative Position Iterator (**Backward**)

```
private class PositionIteratorB implements Iterator<Position<E>> {

  /** A Position of the containing list, initialized to the first position. */
  private Position<E> cursor = last();   // position of the next element to report
  /** A Position of the most recent element reported (if any). */
  private Position<E> recent = null;       // position of last reported element

  /**
   * Tests whether the iterator has a next object.
   * @return true if there are further objects, false otherwise
   */
  public boolean hasNext() { return (cursor != null);  }

  /**
   * Returns the next position in the iterator.
   *
   * @return next position
   * @throws NoSuchElementException if there are no further elements
   */
  public Position<E> next() throws NoSuchElementException {
    if (cursor == null) throw new NoSuchElementException("nothing left");
    recent = cursor;              // element at this position might later be removed
    cursor = before(cursor);
    return recent;
  }

  /**
   * Removes the element returned by most recent call to next.
   * @throws IllegalStateException if next has not yet been called
   * @throws IllegalStateException if remove was already called since recent next
   */
  public void remove() throws IllegalStateException {
    if (recent == null) throw new IllegalStateException("nothing to remove");
    LinkedPositionalList2.this.remove(recent);          // remove from outer list
    recent = null;                // do not allow remove again until next is called
  }
 } //------------ end of nested PositionIteratorB class ------------

 //--------------- nested PositionIterable class ----------------
 private class PositionIterableB implements Iterable<Position<E>> {
   public Iterator<Position<E>> iterator() { return new PositionIteratorB(); }
 } //------------ end of nested PositionIterable class ------------

public Iterable<Position<E>> positionsB() {
    return new PositionIterableB();       // create a new instance of the inner class
  }
```

After → Before

34

Alternative Position Iterator (**Backward**)

```
public class TestListPos2 {



 public static void main(String argv[])
{


 LinkedPositionalList2 listapos= new LinkedPo
 // Iterator<String> iter;
 String[] testo = {"primo","secondo","terzo"}
System.out.println("Avvio lista");


for (String s: testo) {
    listapos.addLast(s);
}
String s1="primonuovo";
listapos.set(listapos.first(),s1);


System.out.println("Dimensione lista= "+ listapos.size());
System.out.println("Primo lista= "+ listapos.first().getElement());
Position p = listapos.after(listapos.first());
System.out.println("Elemento successivo al primo= "+ p.getElement());

Iterator<String> iter=listapos.iterator();

while  (iter.hasNext()) {
    System.out.println("Valore nella lista= "+ iter.next());
 }
 System.out.println();
 Iterable<Position<String>> iter2=listapos.positions();
 for (Position p1: iter2 ) {
    System.out.println("Valore nella lista per posizione = "+ p1.getElement());
 }
System.out.println();
Iterable<Position<String>> iterB=listapos.positionsB();

 for (Position p1: iterB ) {
    System.out.println("Valore nella lista per posizione  a ritroso = "+ p1.getElement());
 }
}
}
```

```
algoritmi miei — -bash — 83×19
Last login: Mon Apr  1 09:54:37 on console
iMac-di-Rocco:algoritmi miei roccoaversa$ java TestListPos2
Avvio lista
Dimensione lista= 3
Primo lista= primonuovo
Elemento successivo al primo= secondo
Valore nella lista= primonuovo
Valore nella lista= secondo
Valore nella lista= terzo

Valore nella lista per posizione = primonuovo
Valore nella lista per posizione = secondo
Valore nella lista per posizione = terzo

Valore nella lista per posizione  a ritroso = terzo
Valore nella lista per posizione  a ritroso = secondo
Valore nella lista per posizione  a ritroso = primonuovo
iMac-di-Rocco:algoritmi miei roccoaversa$
```