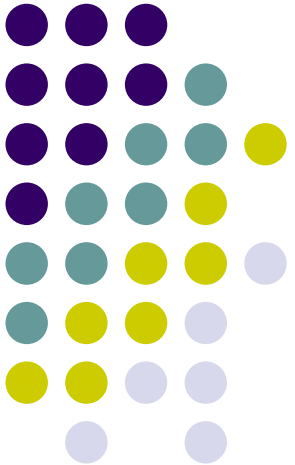
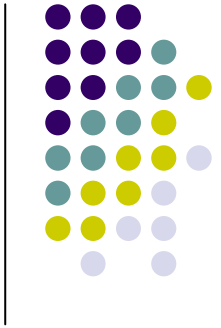
Sequence

Corso: Strutture Dati

Docente: Annalisa De Bonis



Il TDA Sequence

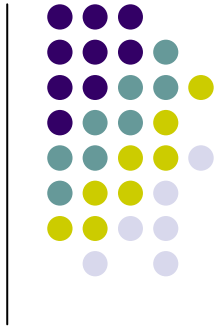


- Il TDA Sequence e` un TDA per la rappresentare un insieme di elementi disposti secondo un ordine lineare
- Fornisce le funzionalita` del TDA Node list e del TDA Array list
  - E` possibile fare riferimento ad un elemento sia attraverso il suo indice che attraverso la sua posizione

Il TDA Sequence

- Oltre ai metodi del TDA Array list e Node list, il TDA Sequence supporta :
- i metodi del Deque e
- due metodi “ponte’’ che mettono in relazione indici e posizioni:
  - atIndex(i)
    - Restituisce la posizione dell’elemento di indice i
  - indexOf(p)
    - Restituisce l’indice dell’elemento in posizione p

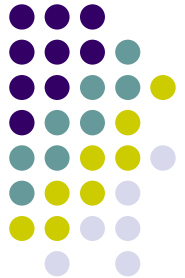
Interfaccia di Sequence



```
public interface Sequence <E> extends
PositionList <E>, IndexList <E>, Deque<E>{
  // Metodi ``ponte''
  public Position <E> atIndex(int index)
    throws BoundaryViolationException;
  public int indexOf(Position <E>position)
     throws InvalidPositionException;
}
```

Ereditarietà multipla

Implementazioni di Sequence



- Lista doppiamente linkata
  - Un nodo contiene un elemento della sequenza

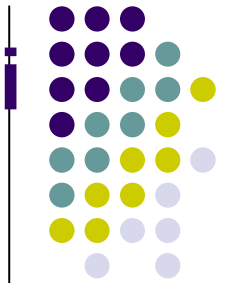
- Array di posizioni
  - Il TDA Position viene implementato in modo da memorizzare, oltre all'elemento, un indice che rappresenta il rango dell'elemento
  - Se l'array e` usato in modo circolare allora i metodi addFirst() e removeFirst() possono essere eseguiti in tempo O(1).

Implementazione basata su lista a doppi puntatori: la classe NodeSequence

```
public class NodeSequence <E> extends
NodePositionList <E>implements Sequence <E> {
// controlla che l'indice sia nel range  [0,n-1]
protected void checkIndex(int index,int n) throws
IndexOutOfBoundsException {
  if (index < 0 || index >= n)
    throw new IndexOutOfBoundsException ("L'indice " +
index + " non e` valido per questa sequenza.");
}
```

Continua nella slide successiva

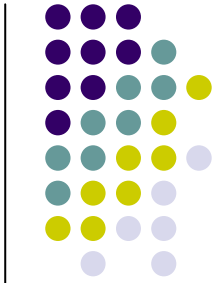
Implementazione basata su lista a doppi puntatori: la classe NodeSequence



```
public Position <E> atIndex (int index)
        throws IndexOutOfBoundsException {

 checkIndex(index,size());
  DNode<E>node;
  if (index <= size()/2) {
      node = header.getNext();
      for (int i=0; i < index; i++)
        node = node.getNext(); } //end if
  else {
       node = trailer.getPrev();
```

Continua nella slide successiva

Implementazione basata su lista a doppi puntatori: la classe NodeSequence



```
//metodi di IndexList
public void add(int index, E element)
      throws IndexOutOfBoundsException {
    if (index == size())
      addLast(element);
    else { checkIndex(index,size());
      addBefore(atIndex(index), element); }
  }
public E remove (int index)
    throws IndexOutOfBoundsException {
  checkIndex(index,size());
```

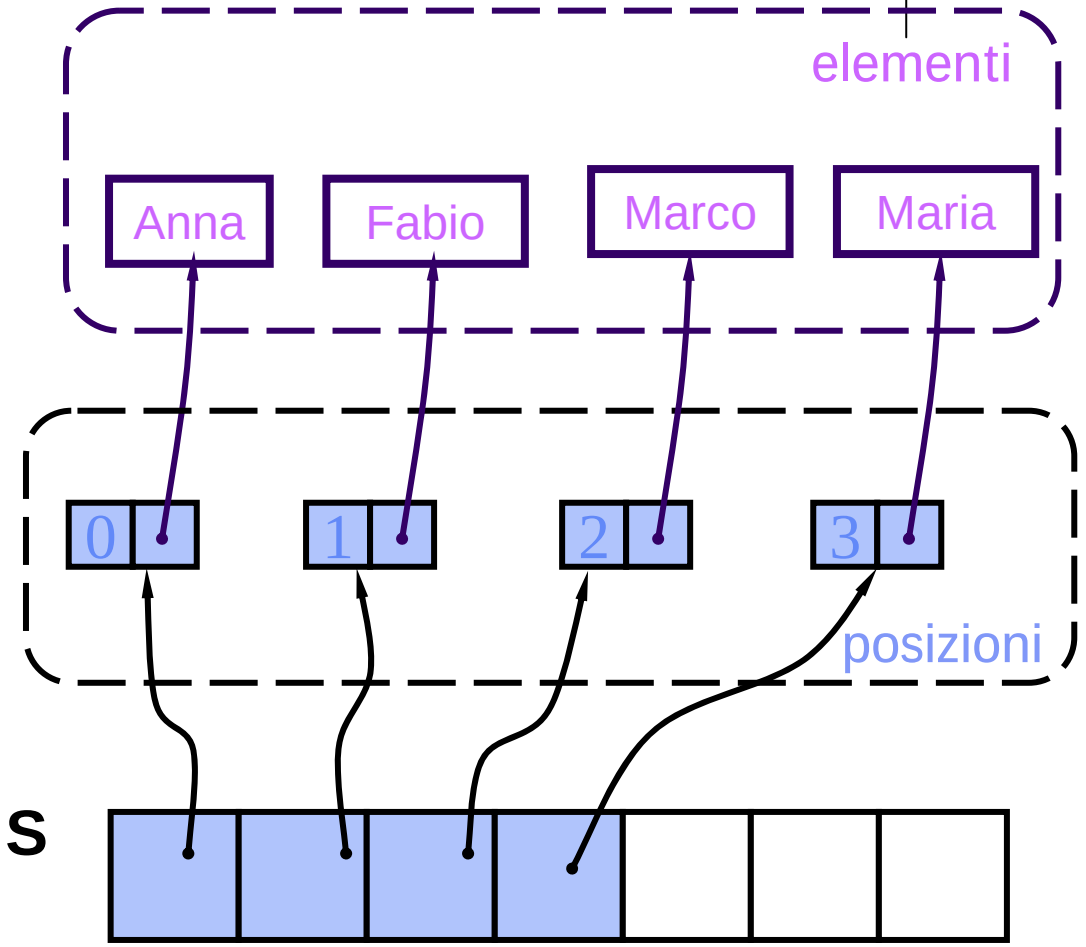
Continua nella slide successiva

Implementazione basata su lista a doppi puntatori: la classe NodeSequence

```
public E set (int index, E element)
      throws IndexOutOfBoundsException {
  checkIndex(index,size());
  return set(atIndex(index),element);
 }
public E get(int index)
      throws IndexOutOfBoundsException {
  checkIndex(index,size());
  return atIndex(index).element();
 }

…
}
```

Implementazione basata su array

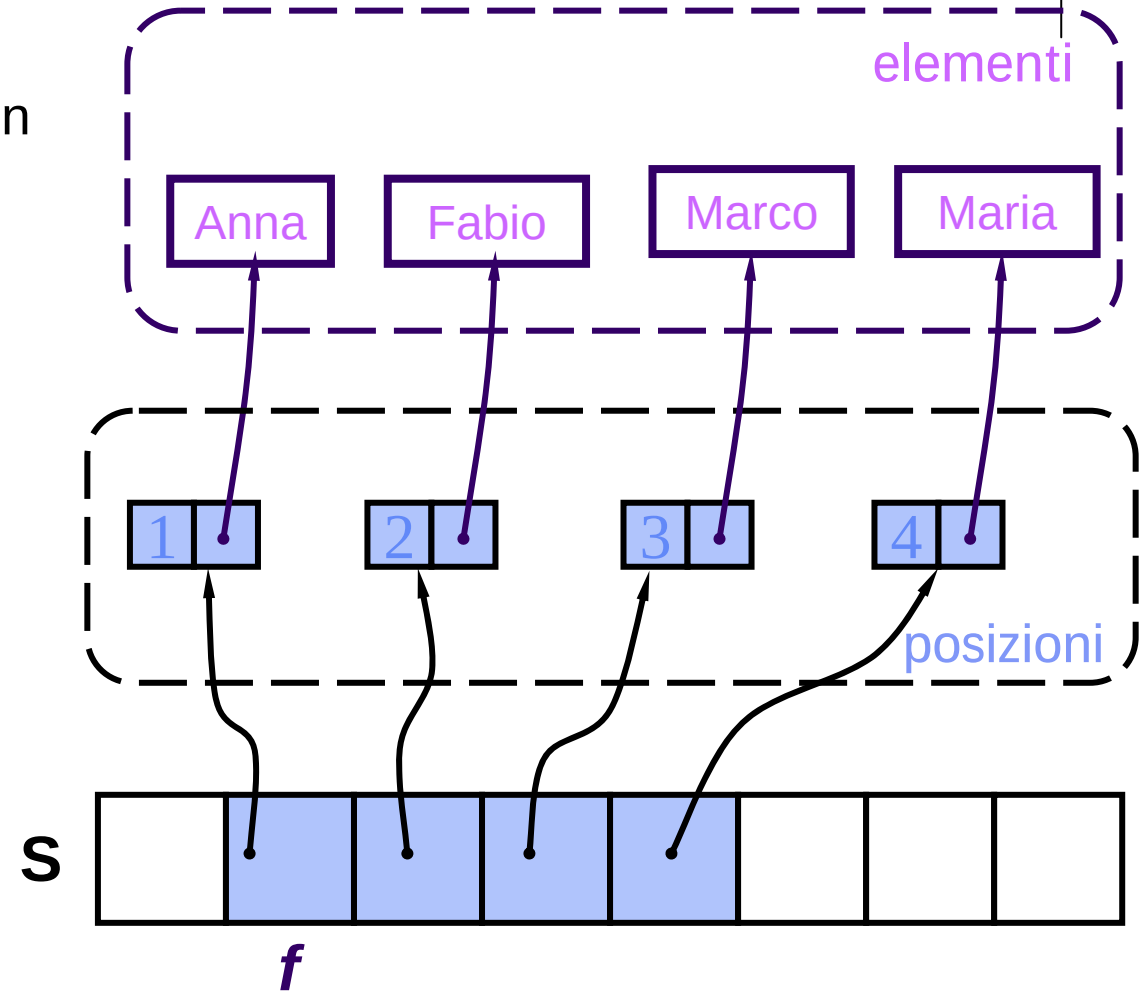
- Ogni locazione contiene un oggetto di tipo Position
- Un oggetto di tipo Position memorizza:
  - Elemento
  - Indice



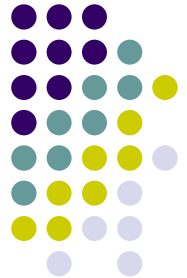
Implementazione basata su array circolare

Un oggetto di tipo Position memorizza:

- Elemento
- Indice della locazione dell'array

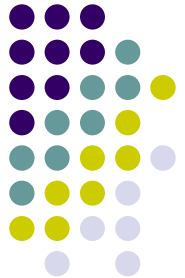


Implementazione basata su Array: la classe ArrayPosition



```
public class ArrayPosition <E> implements Position <E>{
 int index;
 E element;
 public ArrayPosition(int i,E e){index=i; element=e;}
 public ArrayPosition(){}
 public E element(){return element;}
 public int getIndex() {return index;}
 public E setElement(E e){E tmp= element; element=e;return tmp; }
 public int  setIndex(int i){int tmp = index; index=i; return tmp;}
}
```

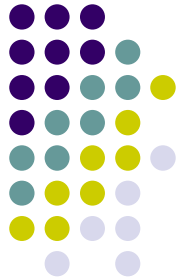
Implementazione basata su Array: la classe ArraySequence



```
public class ArraySequence<E> implements Sequence<E>{

 private ArrayPosition<E>[] A;
  public static final int CAPACITY=1024 ;
  public int capacity ;
  private int size = 0;
```

continua nella prossima slide

Implementazione basata su Array: la classe ArraySequence



```
public Position  <E> atIndex(int index)
  throws IndexOutOfBoundsException {
checkIndex(index,size());
return(A[index]);

}

public int indexOf(Position <E> position)
  throws InvalidPositionException{
ArrayPosition<E> v= checkPosition(position);
return v.getIndex();
}
```

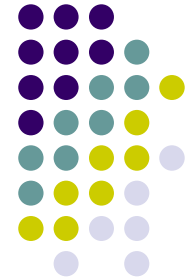
continua nella prossima slide

Implementazione basata su Array: la classe ArraySequence

```
public void add(int r, E e)
   throws IndexOutOfBoundsException {
   checkIndex(r, size() + 1);
   if (size == capacity) {
     capacity *= 2;
     ArrayPosition<E>[] B=(ArrayPosition<E>[]) new ArrayPosition[capacity];
     for (int i=0; i<size; i++)
       B[i] = A[i];
     A = B;    } // end if
   for (int i=size-1; i>=r; i--){
     A[i+1] = A[i];
     A[i+1].setIndex(i+1);    }
     A[r] = new ArrayPosition<E>(r,e);
     size++;
}
```

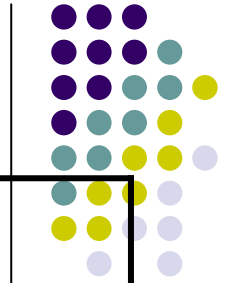
continua nella prossima slide

Implementazione basata su Array: la classe ArraySequence



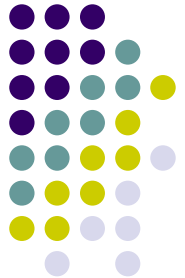
```
public Position<E> addBefore(Position<E> p, E e)
    throws InvalidPositionException {
  checkPosition(p);
  int r= indexOf(p);
  add(r,e);
  return A[r];
 }


…

}
```

Implementazioni di Sequence



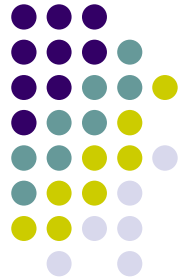
Operazioni	Array	Array Circolare	List
size, isEmpty	1	1	1
atIndex, IndexOf, get	1	1	***n***
first, last, before, after, getFirst, getLast	1	1	1
set che prende in input una posizione	1	1	1
set che prende in input un indice	1	1	***n***
add, remove che prende in input un indice	***n***	***n***	***n***
removeFirst, addFirst	***n***	1	1
addAfter, addBefore	***n***	***n***	1
removeLast, addLast	1	1	1
remove che prende in input una posizione	***n***	***n***	1

Esercizio su Sequence



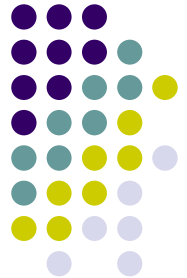
- Completare la classe NodeSequence che estende la classe NodeList e implementa l'interfaccia Sequence mediante una lista doppiamente linkata

Esercizio su Sequence



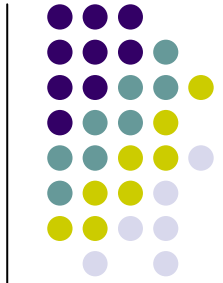
- Scrivere la classe ArraySequence che implementa l'interfaccia Sequence mediante un array.

Esercizio su Sequence



- Scrivere la funzione mergeSort che ordina una sequenza di interi mediante l'algoritmo merge-sort

Esercizio su Sequence



- Scrivere la funzione ricorsiva

  boolean search(Sequence<E>S,E x)

che restituisce true se e solo se la sequenza S contiene l'elemento x.

- Se invocata su $S=\langle s_1,\ldots,s_n\rangle$, la funzione restituisce
  - *false* **se n=0**
  - *true* **se s_1=x**
  - *Il valore resituito da search($\langle s_2,\ldots,s_n\rangle$, x)* **altrimenti**

Esercizio su Sequence

- Aggiungere alla classe il metodo void makeFirst(Position p) che sposta l'elemento in posizione p all'inizio della sequenza lasciando inalterato l'ordine dei rimanenti elementi.