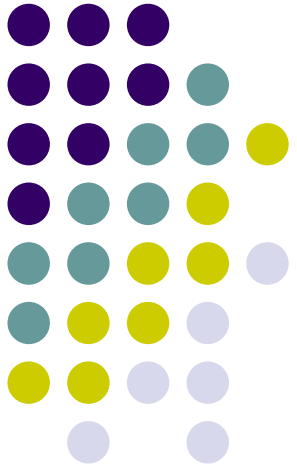
Node List (lista)

Corso: Strutture Dati

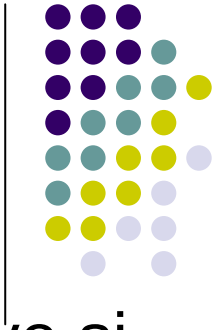
Docente: Annalisa De Bonis



Il TDA Node list

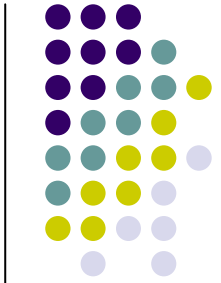
- Analogamente al TDA Array list, il TDA Node list rappresenta una sequenza di elementi disposti secondo un ordine lineare
- Il TDA Node list e` la versione orientata agli oggetti della struttura dati concreta lista a puntatori
- La versione astratta del nodo e` il TDA Position
  - Position nasconde il modo in cui la lista e` implementata

Il TDA Position



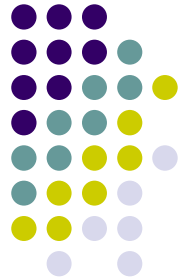
- Il TDA Position modella la nozione di *posto* dove si trova memorizzato un singolo elemento della lista
- Fornisce una visione astratta e unificante dei diversi modi di memorizzare un dato nelle varie implementazione del TDA Node list.
- Ad esempio:
  - una cella di un array
  - un nodo di una lista a puntatori
- Supporta un'unica operazione:
  - element(): restituisce l'elemento memorizzato nel posto rappresentato dall'oggetto di tipo Position

Il TDA Position



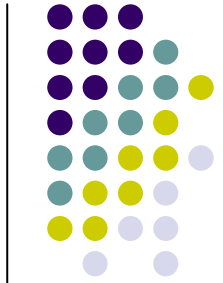
- Una posizione e` definita relativamente ai suoi vicini
  - Eccezion fatta per la prima e l'ultima posizione, ciascuna posizione p si trova sempre prima di una certa posizione q e dopo una certa posizione m
- I metodi che operano sugli elementi della lista hanno come parametri oggetti di tipo Position e restituiscono oggetti di tipo Position
  - Nelle implementazioni del TDA Node list, quali la lista a puntatori singoli o la lista a doppi puntatori, il fatto di usare la posizione dell'elemento (Nodo) invece dell'indice rende i metodi piu` efficienti

L’interfaccia Position



```
public interface Position <E> {
   // Restituisce l’elemento memorizzato
   public E element();
}
```

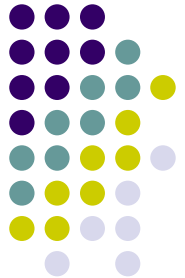
Il TDA Node list



- Collezione che memorizza ciascun elemento in una certa posizione.

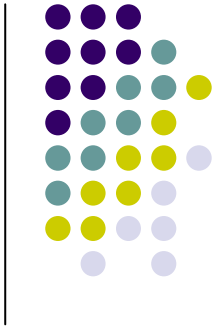
- Stabilisce una relazione prima/dopo tra le posizioni

I metodi del TDA Node list



- Metodi generici
  - size()
    - Restituisce il numero di elementi memorizzati nella lista
  - isEmpty()
    - Retituisce true se e solo se la lista e` vuota

I metodi del TDA Node list

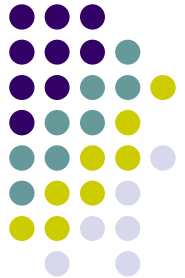


- Metodi di accesso
  - first()
    - Restituisce la posizione del primo elemento della lista . Se la lista e` vuota si verifica un errore.
  - last()
    - Restituisce la posizione dell'ultimo elemento della lista . Se la lista e` vuota si verifica un errore.
  - prev(p)
    - Restituisce la posizione dell' elemento che precede l'elemento in posizione p. Si verifica un errore se p e` la prima posizione.
  - next(p)
    - Restituisce la posizione dell' elemento che segue l'elemento in posizione p. Si verifica un errore se p e` l'ultima posizione.

I metodi del TDA Node list

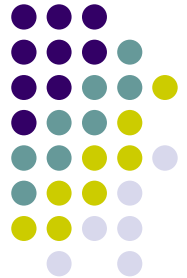
- Metodi di aggiornamento
  - addBefore (p, e)
    - Inserisce l'elemento e nella posizione che precede la posizione p e ne restituisce la posizione
  - addAfter(p, e)
    - Inserisce l'elemento e nella posizione che precede la posizione p e ne restituisce la posizione
  - addFirst(e)
    - Inserisce l'elemento e nella prima posizione della lista
  - addLast(e)
    - Inserisce e nell'ultima posizione della lista
  - remove(p)
    - Rimuove e restituisce l'elemento in posizione p
  - set(p, e)
    - Sostituisce con e l'elemento in posizione p restituendolo in output

Eccezioni di Node list



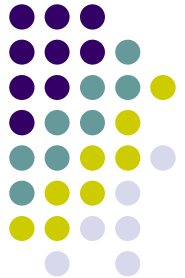
- InvalidPositionException
  - Viene lanciata quando viene specificata una posizione non valida come argomento.
  - Ad esempio:
    - p = **null**
    - p è la posizione di una lista differente
    - p è gia` stata cancellata dalla lista

BoundaryViolationException



- BoundaryViolationException
  - Viene lanciata quando si tenta di accedere ad una posizione fuori della lista
  - Ad esempio:
    - viene lanciata dal metodo prev quando riceve come argomento la prima posizione della lista

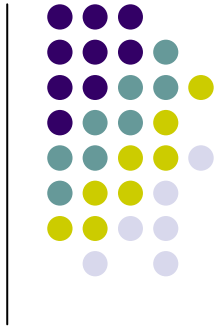
EmptyListException



- Quando i metodi first() e last() vengono invocati su una lista vuota si verifica un errore.
  - Se invocati su una lista vuota, i metodi first() e last() della classe NodePositionList lanciano l'eccezione EmptyListException.

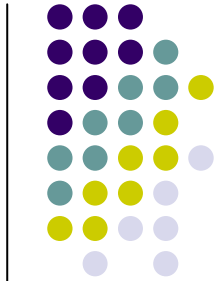
L’interfaccia PositionList

```
public interface PositionList <E>{
   // Metodi generici
      public int size();
      public boolean isEmpty();
```



Continua nella prossima slide

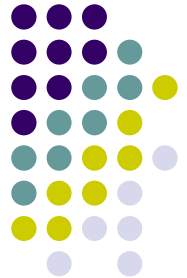
L’interfaccia PositionList



```
// Metodi di aggiornamento
public Position <E> addBefore(Position<E> p, E e)
                    throws InvalidPositionException;

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

public void  addFirst(E e);

public void addLast(E e);

public E remove(Position <E> p)
                    throws InvalidPositionException;

public E set(Position <E> p, E e)
                    throws InvalidPositionException;
```

Continua nella prossima slide

L’interfaccia PositionList



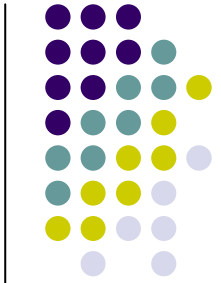
```
// Metodi di accesso
public Position <E> first() throws EmptyListException :
public Position <E> last()throws EmptyListException;

public Position <E> prev(Position <E> p)
                    throws InvalidPositionException,
                                        BoundaryViolationException;

public Position <E> next(Position <E> p)
                    throws InvalidPositionException,
                                        BoundaryViolationException;
}
```

Implementazione di Position

```
public class DNode <E> implements Position <E>{
 // Variabili
   private E element;
   private DNode <E> next,prev;
 // Costruttore
 public DNode(DNode <E> p,E e, DNode <E> n)
  {
     prev = p;
     element = e;
     next = n;
  }
```

Continua nella prossima slide

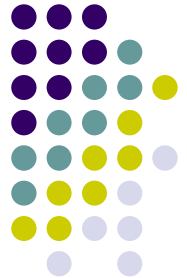
Implementazione di Position



```
// Metodo dell' interfaccia
 public E element() throws InvalidPositionException{
       if ((prev == null) || (next == null))
           throw new InvalidPositionException("Posizione non
                                                valida")
      return element;
 }
// Metodi di accesso
public DNode <E> getNext() { return next; }
public DNode <E> getPrev() { return prev; }
```

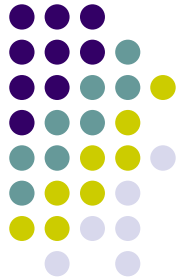
Continua nella prossima slide

L’implementazione di Position



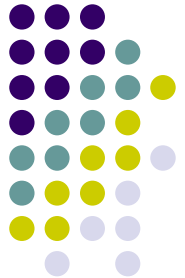
```
// Metodi di aggiornamento
public void setElement(E newElem) {
   element = newElem;
 }
public void setNext(DNode <E> newNext) {
    next = newNext;
 }
public void setPrev(DNode <E> newPrev) {
     prev = newPrev;
 }
}
```

La classe NodePositionList



```
public class NodePositionList<E>
   implements PositionList<E> {

protected int numElts;
protected DNode<E> header, trailer;
  …

}
```

Il costruttore di NodePositionList()

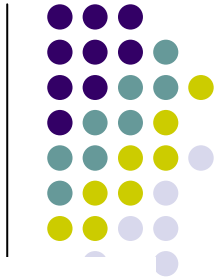


```
public NodePositionList() {
 numElementi = 0;
 header = new DNode(null,null,null);
 trailer = new DNode(header,null,null);
 header.setNext(trailer)
 }
```

Il metodo CheckPosition

- DNode<E> CheckPosition(Position <E> p)
  - Controlla se la posizione p e` valida:
    - se la posizione non e` valida lancia l'eccezione InvalidPositionException
    - altrimenti effettua il casting della posizione passata in DNode

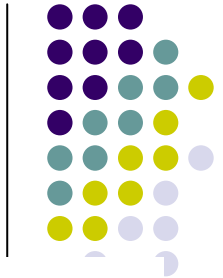
Il metodo CheckPosition



```
DNode <E> CheckPosition(Position<E> p) throws
    InvalidPositionException{
     if (p == null) throw new
    InvalidPositionException("Posizione nulla
                                passata a NodeList");
     if (p == header) throw new
    InvalidPositionException("Header non è
                                una posizione valida");
    if (p == trailer) throw new
    InvalidPositionException("Trailer non è
                                una posizione valida");
```

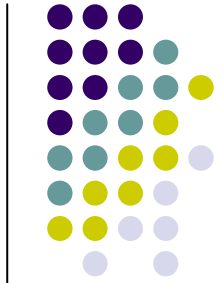
Continua nella prossima slide

Il metodo CheckPosition



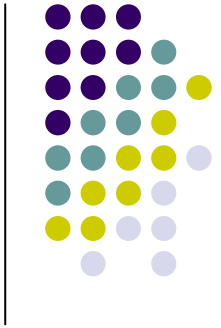
```
try { DNode <E>temp = (DNode<E>) p;       // cast
      if ((temp.getPrev() == null) || (temp.getNext() == null))
         throw new InvalidPositionException("Posizione non
                              appartenente ad una valida NodeList");
        return temp;
   }
  catch (ClassCastException e) {
     throw new InvalidPositionException("Posizione di tipo
    sbagliato per questo contenitore"); }
}
```

Il metodo addBefore



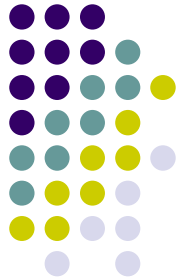
```
public Position<E> addBefore(Position<E> p, E e)
    throws InvalidPositionException {
  DNode<E> v = checkPosition(p);
  numElts++;
  DNode<E> newNode = new DNode<E>(v.getPrev(),v, e);
  v.getPrev().setNext(newNode);
  v.setPrev(newNode);
 return newNode;
}
```

Esercizi



- Implementare l'interfaccia PositionList (scrivere la classe NodePositionList) usando la classe DNode come implementazione dell'interfaccia Position.
- Scrivere un programma che testi tutti i metodi della classe NodePositionList

Esercizi



- Scrivere la funzione

  void removeOdd(PositionList<E> L)

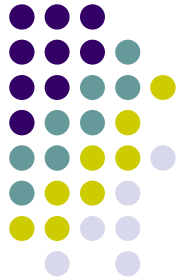
  che rimuove da L tutti gli elementi di rango dispari (il primo, il terzo, ecc.)

- Aggiungere alla classe NodePositionList il metodo

  PositionList <E> creaCopia()

  che restituisce una lista identica a quella su cui il metodo è invocato

Esercizi



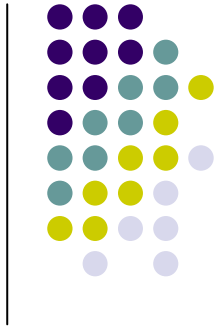
- Scrivere la funzione ricorsiva

  void reverse(PositionList<E> L)

che inverte la lista L.

- Definizione ricorsiva di lista inversa di L=<e_1,e_2,…,e_n >
  - inversa(L)=L se n≤1
  - Inversa(L)= <e_n>+ inversa(<e_1,…,e_{n-1} >) se n>1
    - Indichiamo con ‘+’ l’operatore di concatenazione di 2 liste

Esercizi



- Scrivere la funzione

PositionList<E>merge(PositionList<E>L1,PositionList<E>L2)

che prende in input due liste ordinate L1 ed L2 e restituisce una lista ordinata contenente gli elementi di L1 ed L2.