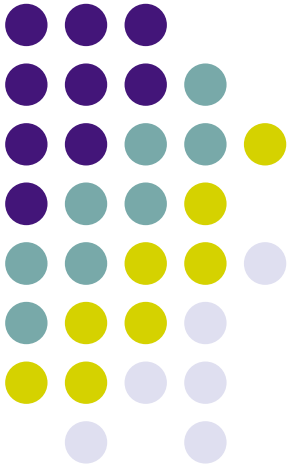
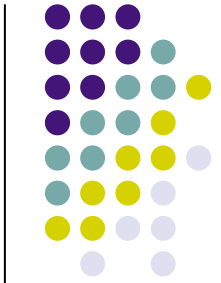
Iterator

Corso: Strutture Dati

Docente: Annalisa De Bonis



Iteratore



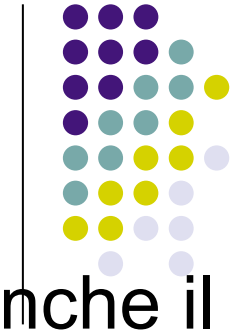
- Il TDA Iterator astrae il processo di scandire gli elementi di un contenitore uno alla volta
- Permette di scandire gli elementi della struttura dati a prescindere dall'implementazione della struttura dati

Il TDA Iterator

- TDA Iterator supporta i seguenti metodi:
  - next() : restituisce il prossimo elemento nell'iteratore
  - hasNext() : testa se ci sono altri elementi nell'iteratore

- Noi implementeremo l'interfaccia java.util.Iterator

java.util.Iterator

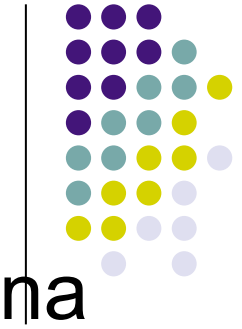


- L'interfaccia Java java.util.Iterator supporta anche il metodo remove che cancella l'elemento precedentemente restituito.

- Le classi che implementano Iterator possono non implementare remove

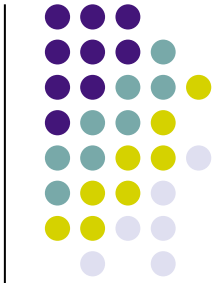
```
public void remove() throws UnsupportedOperationException
 {
    throw new UnsupportedOperationException("remove");
 }
```

Il metodo iterator()



- Un iteratore è tipicamente associato ad una struttura dati che rappresenta una collezione
  - Un iteratore di una sequenza deve restituire gli elementi nell'ordine lineare che hanno nella sequenza
- Possiamo aggiungere ai TDA che rappresentano collezioni il metodo:
  - Iterator <E> iterator()
    - Restituisce un iteratore degli elementi contenuti nella struttura dati

Il tipo Iterable

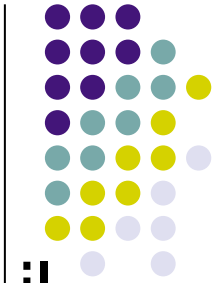


- Le strutture dati che supportano il metodo iterator() estendono la seguente interfaccia

```
public interface Iterable <E>{
  /*Restituisce un iteratore degli elementi
  contenuti nella struttura dati*/
   public Iterator <E> iterator();
}
```

Java fornisce l'interfaccia java.lang.Iterable

Il tipo Iterable



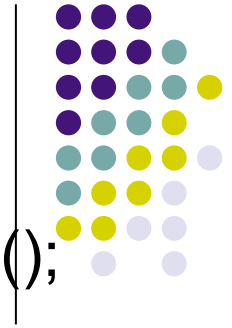
- Se vogliamo che il TDA Node list supporti il metodo iterator() dobbiamo ridefinire l'interfaccia PositionList come segue:

```
public interface  PositionList<E> extends
   Iterable <E>{
  //metodi di PositionList
  …
}
```

Esempio dell'uso degli iteratori

```
PositionList <Integer> L=new NodePositionList<Integer>();
int somma=0;
… //istruzioni che inseriscono un certo numero di elementi in L
Iterator <Integer> it = L.iterator();
while(it.hasNext()){
   int val =  it.next();
   somma=somma+val;
}
```

Se non avessimo usato gli iteratori



```
PositionList <Integer> L=new NodePositionList<Integer>();
int somma=0;
… //istruzioni che inseriscono un certo numero di elementi in L
If(!L.isEmpty()){
   Position <Integer> p = L.first();
   while(p!=L.last()){
    int val =  p.element();
    somma=somma+val;
   p=p.next();
  }
   somma=somma+p.element() //sommiamo qui l'ultimo elemento
                  //per non invocare  next() sull'ultima posizione
}
```

Il metodo positions()

- Nei TDA che supportano la nozione di Position possiamo aggiungere il metodo:
  - Iterable <Position <E>> positions()
    - Restituisce un oggetto di tipo Iterable (collezione iterabile) contenente le posizioni del TDA come elementi
  - Se invochiamo iterator() sulla collezione restituita da positions() otteniamo un iteratore delle posizioni del TDA

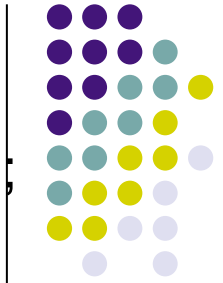
Il metodo positions()

- Se vogliamo che il TDA Node list supporti il metodo positions() dobbiamo ridefinire l'interfaccia PositionList come segue:

```
public interface  PositionList<E> extends Iterable <E>{
 public Iterable <Position <E>> positions();

 //metodi di PositionList
  …
}
```

Costrutto abbreviato

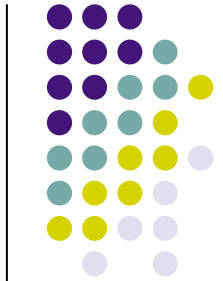


```
PositionList<Integer> values=new NodePositionList<Integer>();
//istruzioni per inserire interi nella lista
…
int sum = 0;
Iterator<Integer> it = values.iterator();
  while(it.hasNext())
  sum+=it.next();
  }
```

È la stessa cosa di

```
PositionList<Integer> values=new NodePositionList<Integer>();
//istruzioni per inserire interi nella lista
…
int sum = 0;
for (Integer i : values)
   sum += i;
```

Il metodo positions() di NodePositionList



```
public Iterable <Position<E>> positions(){
// toReturn è una lista i cui elementi sono di tipo Position
PositionList <Position<E>> toReturn = new NodePositionList
    <Position<E>>();
if(!isEmpty()){
 Position <E> current=first();
 for(int i=0;i<size()-1;i++){
      toReturn.addLast(current);
      current=next(current);
  }
 toReturn.addLast(last()); }   //fine if
 return (toReturn);
 }
```

Implementare un iteratore mediante una struttura dati

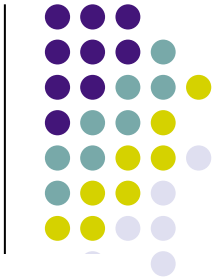
- Possiamo utilizzare qualsiasi struttura dati che consente l'accesso sequenziale ai suoi elementi:
  - In genere si utilizza una Lista linkata o un Array
- Per creare un iteratore bisogna copiare tutti gli elementi della struttura dati che si vuole scandire nella struttura dati usata per implementare l'iteratore
  - iterator() richiede tempo O(n) (n=numero elementi)

L'eccezione NoSuchElementException

- Se non ci sono altri elementi da scandire il metodo next() lancia l'eccezione NoSuchElementException.

- Java fornisce l'eccezione java.util.NoSuchElementException

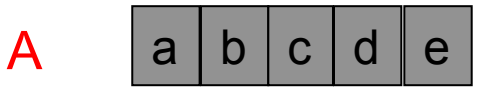
Implementazione mediante lista linkata



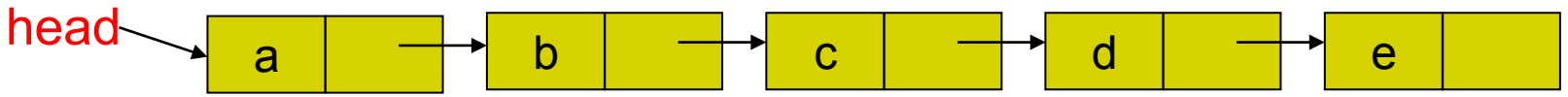
Si copiano gli elementi in una lista linkata

- Qui usiamo una lista a puntato singoli
- La classe che implementa Iterator in questo modo si chiama LinkedIterator

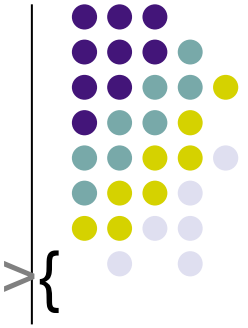
- Esempio:
  - Vogliamo scandire la sequenza <a,b,c,d,e,f>
  - Prima di invocare il costruttore copiamo gli elementi in una array A



  - Il costruttore prende in input l'array e copia gli elementi in una lista



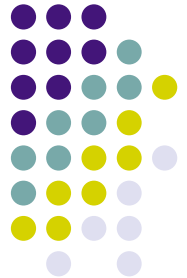
Implementazione mediante lista linkata



```
public class LinkedIterator <E> implements Iterator<E>{
        private Node <E> head;
        private int size;
    //il costruttore prende in input un array contenente
    //gli elementi della collezione
        public LinkedIterator(E [ ] A){
           head=null;
           size=A.length;
           for(int i=size-1;i>=0;i--){
                Node<E> tmp= new Node(A[i],head);
                 head= tmp;
           }
```

Continua nella slide successiva

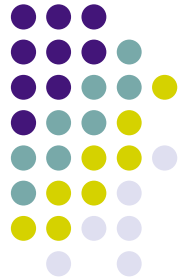
Implementazione mediante lista linkata



```
public boolean hasNext() {
            return ( size >0) );
}

public E next() throws NoSuchElementException{
   if(!hasNext())
        throw new NoSuchElementException("Nessun altro
    elemento");
   E toReturn = head.getElement();
   head=head.getNext();
   size--;
  return toReturn;
}
```

Il metodo iterator() di NodePositionList

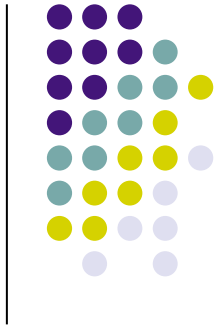


```
public Iterator<E> iterator(){
E []temp= (E[ ])new Object[size()];
//copia gli elementi nell'array da passare al costruttore di //
    LinkedIterator
  if(!isEmpty()){
   Position<E> current=first();
    for(int i=0;i<size()-1;i++){
      temp[i]=current.element();
      current=next(current);
}
    temp[size()-1]=last().element();
    }  //fine if
 return (new LinkedIterator(temp));
  }
```

Implementare un iteratore mediante un cursore

- In questa implementazione l'iteratore opera sulla collezione che si vuole scandire invece che su una sua copia
- Si usano
  - Una variabile che fa riferimento alla collezione di elementi
  - Una variabile di istanza che tiene traccia dell'elemento corrente (cursore)
- Per creare un iteratore bisogna semplicemente inizializzare le due variabili di istanza
  - iterator() richiede tempo O(1)
- È un iteratore *semi-dinamico* in quanto tiene conto di alcune modifiche che vengono effettuate sulla lista

La classe ElementIterator per PositionList

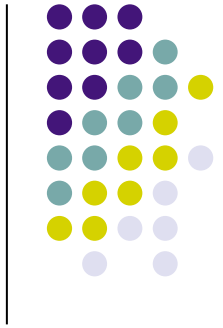


```
public ElementIterator<E> implements Iterator<E>{
protected PositionList<E> list;  //riferimento alla lista da scandire
protected Position<E> cur = null; //cursore
public ElementIterator(){   } //crea iteratore su collezione vuota

public ElementIterator(PositionList<E> L) {
    list = L;
    if (!list.isEmpty())
      cur = list.first(); //inizializzazione del cursore
  }
```

Continua nella slide successiva

La classe ElementIterator per PositionList



```
public boolean hasNext() { return (cur != null); }

public E next() throws NoSuchElementException {
 if (!hasNext()) throw new NoSuchElementException("nessun'altra
    posizione");
 E toReturn = cur.element();
 if (cur == list.last()) cur = null;  // siamo arrivati alla fine della  lista
 else cur = list.next(cur);          // sposta il cursore
 return toReturn;
  }
}
```

Il metodo iterator() di List

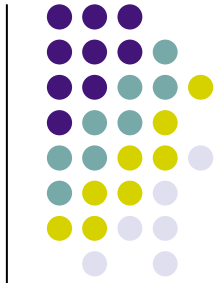
- Usando l'implementazione precedente di ElementIterator è semplicissimo scrivere il metodo iterator() di una classe che implementa PositionList

```
public Iterator iterator() {
   return new ElementIterator(this);
 }
```

Impl. mediante struttura dati vs impl. con cursore

- Implementazione mediante struttura dati
  - Vantaggio: la classi che implementano Iterator in questo modo permettono di creare iteratori per una qualsiasi struttura dati
  - Svantaggio: la creazione di un iteratore richiede tempo lineare nel numero di elementi della struttura dati da scandire
- Implementazione mediante cursore
  - Vantaggio: la creazione di un iteratore richiede tempo costante
  - Svantaggio: per ciascuna struttura dati che vogliamo scandire dobbiamo fornire un'apposita classe che implementa Iterator per quella struttura dati

Esercizi



- Scrivere la classe IndexListIterator che implementa Iterator mediante un vettore

- Scrivere un programma che stampa gli elementi di una lista mediante un iteratore.