Corso sul linguaggio Java

Modulo JAVA7

B1–Gestione di liste

Prerequisiti

- Programmazione base in Java
- Utilizzo di classi e oggetti
- Tecnica di allocazione dinamica
- Gestione di liste
- Classificazione delle liste

Introduzione

In questa Unità vediamo come realizzare in pratica applicazioni che gestiscono liste concatenate.

Si descrive, infine, la classe **LinkedList** di Java, che implementa tutte le funzionalità di una lista.

Strutture dati concrete e astratte

In base al tipo di problema, il programmatore deve scegliere il tipo di struttura dati adatta a rappresentare la realtà di interesse.

- **Strutture dati concrete**:
  - array
  - array dinamico
  - matrice
- **Strutture dati astratte**:
  - lista
  - sequenza
  - pila
  - coda
  - albero
  - grafo
  - tabella

Array dinamico e liste

In generale, sappiamo che *quando non si conosce a priori il numero di oggetti che il programma dovrà gestire*, possiamo utilizzare la tecnica dell'allocazione dinamica, attraverso:

- **Array dinamico**: (v. Unità precedente) se è richiesto che la struttura dati debba avere *sia sequenzialità fisica* (allocazione in locazioni di memoria contigue) *che sequenzialità logica* (possibilità di accedere alle celle mediante un indice);
- **Lista concatenata**: qualora la *sequenzialità fisica non sia richiesta*. In questo caso, spetta al programmatore gestire i collegamenti (riferimenti) tra le celle consecutive.

Implementazione di una lista

PROBLEMA

Si acquisisce da input una serie di nomi di persona che vanno memorizzati in una struttura dati. Sui dati occorre prevedere almeno la possibilità di inserimento in testa e in coda, di ricerca e di stampa di tutti i nomi.

ATTIVITA': sviluppare l'analisi del problema.

Per creare la lista, sono necessarie due classi: la classe *node* e la classe *list*. Supponiamo che la classe *node* abbia due soli attributi: *info*, per contenere l'informazione e *next* per contenere il riferimento alla cella successiva. Inizialmente, forniamo alla classe *node* i metodi di default. La classe *list* prevede un solo attributo, diciamo *head*, che indica il riferimento di accesso alla lista, e i classici metodi di default.

Implementazione di una lista

ATTIVITA': utilizzando l'analisi prodotta, tracciare la UML delle classi necessarie *node* e *list*.

La UML delle classi utilizzate è la seguente

node
-- **Stringa** info; -- node next;
+ node(); + node(**Stringa** str); + **String** getInfo(); + **void** setInfo(**String** x); + node getNext(); + **void** setNext(node next);

list
- node head;
+ list(); + **void** printList(); + **void** insTesta (**Stringa** str); + **void** insCoda (**Stringa** str); + **void** insInterno (**Stringa** str); + **void** eliTesta (); + **void** eliCoda (); + **void** eliInterno (); + node contains (**Stringa** str); + **boolean** contained(**Stringa** str); + **void** menu (); + **void** opera (**Intero** s);

La classe *node*

ATTIVITA': implementare la classe *node*

```
import java.io.*;
public class node
{   private String info;
    private node next;
    public node()
    {   info="";   next = null;   }
    public node (String str)
    {   info = str;  next = null;   }
    public String getInfo ()  { return info;  }
    public void setInfo (String str)   {  info = str; }
    public node getNext ()    { return next;  }
    public void setNext (node next)   { this.next = next;  }
}   /* fine classe */
```

La classe *testnode*

ATTIVITA': implementare la classe *nodetest.java*.

```
import java.io.*;
{ ….
    String nome;
    node p;
    nome=Tastiera.readLine();
    creare tre celle di prova c1, c2 e c3
    stampare i dati presenti nelle 3 celle
    collegare c1 con c2 e c2 con c3
    usando il solo accesso a c1, stampare Info(c1), Info(c2) e Info(c3)
    ….
}        /* fine classe */
```

Vediamo nello specifico la codifica di queste istanze lasciate in sospeso.

La classe *testnode*

creare tre celle di prova c1, c2 e c3

ATTIVITA': creare *c1 e c3* con costruttore con parametro *nome*; creare *c2* con costruttore senza parametri e successivamente impostare *Info(c2)* con *Info(c1)*.

```
node c1=new node(nome);
node c2=new node();
node c3=new node(nome);
c2.setInfo(nome);
```

La classe *testnode*

stampare i dati presenti nelle tre celle

ATTIVITA': istanziare il metodo *getInfo*() per stampare i campi *Info(c1)*, *Info(c2)* e *Info(c3)*.

```
System.out.println("Info(c1) =" + c1.getInfo());
System.out.println("Info(c2) =" + c2.getInfo());
System.out.println("Info(c3) =" + c3.getInfo());
```


La classe *testnode*

collegare c1 con c2 e c2 con c3

ATTIVITA': istanziare il metodo *setNext*() per collegare *c2* di seguito a *c1* e *c3* di seguito a *c2*.

```
c1.setNext(c2);
c2.setNext(c3);
```

La classe *testnode*

usando il solo accesso a c1, stampare Info(c1), Info(c2) e Info(c3)

ATTIVITA': inizializzare un riferimento *p* con *c1* e tramite un ciclo **for** avanzare *p*, stampando, di volta in volta, *Info(p)*.

```
p=c1;
for (int i=0; i<3; i++)
{
    System.out.println("i="+ i + ": " + p.getInfo());
    p=p.getNext();
}
```


La classe *list*

ATTIVITA': implementare la classe *list*

```
import java.io.*;
public class list
{   private node head;
    public list()   { head = null;  }
    public void insTesta(String str)  {….}
    public void printList ()  { …}
    public void insCoda (String str)  { …}
    public boolean contained(String str)  { …}
    public static void main(String args[])
    {       list l = new list();
            inserimento in l di alcune celle;
            verifica della presenza di un valore;
            stampa della lista;
    }
}   /* fine classe */
```

Vediamo nello specifico la codifica di queste istanze lasciate in sospeso.

La classe *list*

inserimento in l di alcune celle

ATTIVITA': scrivere nella lista *l* alcune *stringhe*. Inserire in testa "Anna", "Mario" e "Carla". Inserire in coda "Franco"

```
l.insTesta ("Anna");
l.insTesta ("Mario");
l.insTesta ("Carla");
l.append ("Franco");
```


La classe *list*

verifica della presenza di un dato valore

ATTIVITA': scrivere l'istanza del metodo *contained* ("libro") per vedere se è presente la stringa "libro".

```
System.out.println (l.contained ("libro"));
```

La classe *list*

stampa della lista

ATTIVITA': istanziare il metodo *printList*() sulla lista *l*.

```
l.printList();
```


Il metodo printList()

ATTIVITA': implementare il metodo *printList*().

```
public void printList ()
{
    node c = head;                    // dichiara e inizializza oggetto c
    while (c! = null)                 // fintantochè c non è arrivato alla fine…
    {   System.out.println (c.getInfo());   //…stampa Info(c) e…
        c = c.getNext();                    //…avanza al successivo
    }
}
```

Il metodo insTesta(String str)

ATTIVITA': implementare il metodo *insTesta*().

```
public void insTesta (String str)
{
    node c = new node(str);          // crea oggetto c
    c.setNext(head);                 // collega c in testa a head
    head = c;                        // aggiorna head con c
}
```


Il metodo insCoda(String str)

ATTIVITA': implementare il metodo *insCoda*().

```
public void insCoda (String str)
{
    node c = head;                   // crea oggetto c di lavoro
    if (c = = null)                  // se la lista è vuota…
        insTesta(str);               //… inserisci in testa
    else                             //… altrimenti…
    {   while(c.getNext()! = null)   //… fintantochè c non è l'ultimo…
            c=c.getNext();           //… avanza al successivo
        node n = new node(str);      // Crea nuovo nodo in n e…
        c.setNext(n);                //… collega n di seguito a c
    }
}
```

Il metodo insInterno(…)

```
public void insInterno (String str, int pos)
{   node prec, succ;
    int i=0;
    if ((pos==0) || (head==null))   insTesta(str);  // se vuota, inserisci in testa
    else    if (pos>lunghezza())    insCoda(str);  // se pos eccede, inserisci in coda
            else  {   prec=head;                   // inizializza riferimenti
                      succ=head.getNext();
                      while (i<pos-1)              // cerca posizione precedente …
                     {  prec=succ;                 // … a quella dove inserire
                        succ=succ.getNext();
                        i++;     }
                     node n = new node(str);       // crea cella
                     prec.setNext(n);              // … collegala alla precedente
                     n.setNext(succ);       }      // … collegala alla successiva
}
```

ATTIVITA': implementare il metodo *insInterno*().

Il metodo eliTesta()

```
public void eliTesta()
    {
        if (head! = null)
        {
                node c = head;
                head = head.getNext();
        }
    }
```

ATTIVITA': implementare il metodo *eliTesta*().

Il metodo eliCoda()

ATTIVITA': implementare il metodo *eliCoda*().

```
public void eliCoda()
{   if (head! = null)
    {       if (lunghezza()==1)   eliTesta();
            else        {           node prec = head;
                                    node succ = head.getNext();
                                    while (succ.getNext()!=null)
                                    {           prec = succ;
                                                succ = succ.getNext();
                                    }
                                    prec.setNext (null);
                        }
    }
}
```


Il metodo eliInterno()

Il metodo *eliInterno* (**Intero** pos) è lasciato per esercizio al lettore.

Il metodo contains(String str)

ATTIVITA': implementare il metodo *contains* (**String** str).

```
public node contains (String str)
{
    node c=head;
    while((c!=null) && (!c.getInfo().equals(str)))
        c=c.getNext();
    return (c);
}
```


Il metodo contained(String str)

ATTIVITA': implementare il metodo *contained* (**String** str).

```
public boolean contained (String str)
{
    node c = head;
    while((c!= null) && (!c.getInfo().equals (str)))
        c = c.getNext();
    return (c!=null);
}
```

La classe LinkedList di Java

Java fornisce la classe **LinkedList** che implementa tutte le caratteristiche di una struttura a lista.

In particolare, i metodi forniti consentono di navigare sulla struttura in entrambi i versi, per cui un oggetto di classe **LinkedList** può essere immaginato come una *lista doppia*, o *bidirezionale*.

La classe LinkedList di Java

METODO	EFFETTO
LinkedList()	Crea una lista vuota
void add (**Object** *elem*)	Inserisce *elem* in coda alla lista
void addLast (**Object** *elem*)	Inserisce *elem* in coda alla lista
void add (**int** index, **Object** *elem*)	Inserisce *elem* in posizione *index* nella lista
void addFirst (**Object** *elem*)	Inserisce *elem* in testa alla lista
void clear ()	Rimuove tutti gli elementi dalla lista
boolean contains (**Object** *elem*)	Dà **true** se elem esiste nella lista
Object element ()	Dà il primo elemento della lista senza rimuoverlo
Object get (**int** *index*)	Restituisce l'elemento in posizione *index*
Object getFirst ()	Restituisce l'elemento in prima posizione
Object getLast ()	Restituisce l'elemento in ultima posizione
int IndexOf (**Object** *elem*)	Dà la posizione della prima occorrenza di *elem*
Object remove ()	Dà il primo elemento della lista e lo rimuove
Object removeFirst ()	Dà il primo elemento della lista e lo rimuove
Object remove (**int** *index*)	Dà l'elemento in posizione *index* e lo rimuove
Object removeLast ()	Dà l'ultimo elemento della lista e lo rimuove
Object set (**int** *index*, **Object** *elem*))	Sostituisce con *elem*, l'elemento in posizione *index*
int size ()	Dà il numero di elementi presenti

Esempi di LinkedList

```
import java.util.*;
public class lista
{   public static void main(String args[])
    {       LinkedList<String> l = new LinkedList<String>();
            l.addFirst("Mauro");            // inserimento in testa
            l.add("Anna");                  // inserimento in coda
            l.add(1, "Ugo");                // inserimento intermedio
            System.out.println(l);          // stampa la lista
            System.out.println("Rimuovo la testa");
            l.remove();                     // eliminazione in testa
            System.out.println(l);
            System.out.println(l.contains("Ugo"));
    }
} //end class
```


Esempi di LinkedList

```
import java.util.*;
public class lista1
{
   public static void main (String[] args)
   {Object elenco[] = new Object[]{"dog", "cat"};
    LinkedList<Object> list =
            new LinkedList<Object> (Arrays.asList(elenco));
    System.out.println(list);               // [dog, cat]
    System.out.println( list.getFirst() );  // dog
    System.out.println( list.getLast() );   // cat
    System.out.println (list.removeFirst(); // dog
   System.out.println(list);                // [cat]
   }
}
```

Un modo per precaricare nella lista un elenco di valori.

Argomenti

- Strutture dati concrete e astratte
- Array dinamico e liste
- Implementazione di una lista
- La classe *node*
- La classe *testnode*
- La classe *list*
- Il metodo printList()
- Il metodo insTesta (**String** str)
- Il metodo insCoda (**String** str)
- Il metodo insInterno (**String** str, **Intero** pos)
- Il metodo eliTesta ()
- Il metodo eliCoda ()
- Il metodo eliInterno (**Intero** pos)
- Il metodo contains (**String** str)
- Il metodo contained (**String** str)
- La classe **LinkedList** di Java
- Esempi di **LinkedList**