S.I.S. Piemonte

Corso abilitante speciale
Legge 143/ter D.M. 85/2005

Classe di Concorso
A042 – Informatica

MICROCORSO

PROGRAMMAZIONE A

Prof. Domenico Valle

Relazione: UNA LEZIONE DI PROGRAMMAZIONE IN JAVA

Argomento: **ORDINAMENTO DI UN VETTORE TRAMITE L'ALGORITMO *BUBBLE SORT***

Autore: ERIK AMEDEO VIOTTI

20 dicembre 2006

COLLOCAZIONE CURRICOLARE

La lezione è rivolta ad una classe terza (ultimo periodo didattico) di un Istituto Tecnico Industriale con indirizzo Informatico.
Si presuppone una lezione di tre ore, suddivise tra aula e laboratorio.

PREREQUISITI RICHIESTI

Fondamenti teorici della programmazione.
Conoscenze di base del linguaggio Java, in particolare:

- uso delle istruzioni di iterazione e selezione;
- tipi di dati primitivi e *array*;
- gestione dell'*input/output*;
- concetto di classe;
- concetto di metodo di tipo *static*;
- scrittura dei commenti per la documentazione.

Padronanza di un editor Java in laboratorio.

L'esercitazione in laboratorio non fa uso degli oggetti Java, rappresentando un esempio di utilizzo dei metodi *static*. La capacità di usare gli oggetti è quindi un prerequisito non indispensabile.

OBIETTIVI

Comprendere il concetto di ordinamento.
Definire l'algoritmo *bubble sort* con variabile *flag*.
Essere in grado di implementare l'algoritmo nel linguaggio Java, utilizzando le classi. Esempio di utilizzo dei metodi *static*.

ORGANIZZAZIONE DELLE ATTIVITA'

Argomenti teorici tramite lezione frontale in aula; applicazione pratica in laboratorio tramite lavoro a coppie.

MATERIALI E STRUMENTI

Libro di testo di riferimento.
Laboratorio con numero macchine almeno pari alla metà degli studenti.
Software: ambiente di sviluppo appropriato (Java Virtual Machine e editor Java).

LEZIONE FRONTALE (IN AULA)

Supponiamo di avere un vettore di numeri interi. Questo vettore si può dire “ordinato” se ognuno dei suoi elementi è minore o uguale a quello successivo. Del resto la parola “ordinato” suggerisce proprio questo concetto, e siamo abituati ad usarla sia applicata ai numeri che alle lettere.

Per esempio, osserviamo questi sei elementi:

32	75	74	32	2	90

Come si vede non sono ordinati. Invece lo sono disposti così:

2	32	32	74	75	90

Ora cerchiamo di affrontare questo problema:

Scrivere un programma che,
preso in input un vettore,
lo restituisca in output ordinato.

Per prima cosa occorre pensare ad un algoritmo appropriato, il che è come sempre il primo problema del programmatore.

FASE DI DIALOGO CON LA CLASSE

Molto spesso succede di bloccarsi proprio a questo punto, ed è probabile che nessuno degli studenti riesca a risolvere il problema in poco tempo. Eppure, osservando i sei numeri di esempio, tutti sarebbero in grado di ordinarli a mente e in pochi secondi. Questo significa che l’algoritmo risolutivo è in realtà conosciuto da tutti. La parte difficile quindi non è trovare l’algoritmo, bensì trovare il modo di “esprimerlo”, definirlo con precisione.
Ogni studente cerchi di spiegare che cosa fa per ordinare quei sei numeri. E’ probabile che il metodo più usato sia quello di ricercare il numero minore e porlo nella prima posizione, quindi cercare il minore tra i rimanenti e porlo in seconda posizione e così via. Si può notare come il metodo funzioni ma non sia efficiente in termini di numero di operazioni da effettuare. Per convincere gli studenti di ciò basta supporre che il vettore non sia di sei elementi ma di *seimila* elementi.

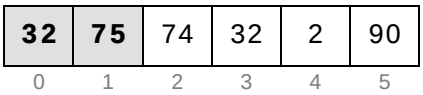
Immaginiamo una fila di persone, e di doverla ordinare per altezza. Di solito si procede scambiando le persone fino a che non risultano ordinate.
Questo metodo è in informatica uno dei molti algoritmi di ordinamento e prende il nome di **BUBBLE SORT**. (NOTA: questo algoritmo in realtà non è molto efficiente per numero di operazioni, ma è abbastanza semplice ed istruttivo)
Si tratta di scambi successivi eseguiti in una struttura iterativa, e per capirne il funzionamento lo applicheremo ai soliti sei numeri.

Ecco il vettore (chiamiamolo **v**) nella sua situazione iniziale (da ora in poi verranno anche indicati gli indici degli elementi):

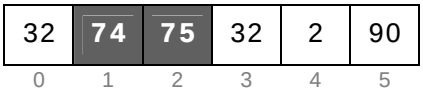
32	75	74	32	2	90
0	1	2	3	4	5

Si confrontano tutte le coppie di elementi adiacenti (il primo e il secondo, il secondo e il terzo, il terzo e il quarto e così via), e nel caso in cui l'elemento a sinistra sia più grande di quello a destra si effettua lo scambio.

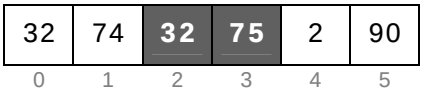
v[0] **non è** maggiore di v[1], per cui **non si fa** lo scambio:



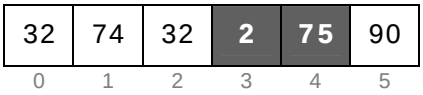
v[1] **è** maggiore di v[2], per cui **si fa** lo scambio:



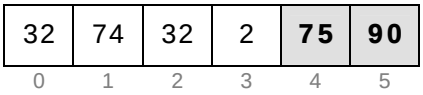
v[2] **è** maggiore di v[3], per cui **si fa** lo scambio:



v[3] **è** maggiore di v[4], per cui **si fa** lo scambio:



v[4] **non è** maggiore di v[5], per cui **non si fa** lo scambio:



Osservando il numero 75 nelle varie fasi, lo si vede muoversi verso la parte finale del vettore come una bollicina in un bicchiere di spumante... ecco da dove viene lo strano nome di questo algoritmo!
Ora che abbiamo terminato di confrontare le coppie, si può notare che il vettore non è ancora ordinato. E' solo un po' meno disordinato, per esempio il 2 si deve ancora spostare a sinistra, come il 74 a destra...
Occorre allora ripetere le stesse operazioni di confronto. Ma quante volte?
A questa domanda si può rispondere in due modi:

Numero fisso di volte

Calcoliamo il numero massimo di confronti che potrebbero servire, immaginando il caso limite in cui il numero più piccolo si trovi proprio al fondo del nostro vettore, nell'elemento 5; in questo caso questo elemento dovrebbe effettuare 5 "salti" per poter arrivare all'elemento 0, cioè all'inizio del vettore. E per fare un "salto" verso sinistra serve un intero ciclo di confronti.
In generale, se il vettore ha n elementi, occorreranno n-1 cicli di confronti per essere certi che il vettore sia ordinato.
Essendo ogni ciclo composto a sua volta di n-1 confronti, il totale dei confronti sarà

$$(n-1)*(n-1).$$

Numero variabile di volte

In realtà la soluzione appena vista è abbastanza semplice, ma poco efficiente. Infatti n-1 cicli sono necessari, come abbiamo visto, solo nel caso limite, mentre sono assolutamente troppi nel caso in cui il vettore sia già in parte ordinato. Meglio sarebbe continuare a effettuare cicli di scambi solo fino a quando il vettore risulta ordinato.
Per fare questo basta verificare ad ogni ciclo se vengono effettivamente eseguiti degli scambi di elementi. Quando ciò non avviene significa che il vettore è già ordinato e l'algoritmo può terminare senza effettuare ulteriori inutili confronti.
Naturalmente resta il fatto che al massimo verranno comunque effettuati n-1 cicli.

Questo metodo si chiama **BUBBLE SORT CON FLAG DI CONTROLLO**, ed è proprio quello che implementeremo in laboratorio.

FASE DI DIALOGO CON LA CLASSE

Per apprezzare l'efficienza di questo secondo metodo basta pensare a grandi numeri: se avessimo un vettore di seimila elementi disordinato per una sola coppia, con il primo metodo verrebbero effettuati

(6000-1)*(6000-1) = 35.988.001

confronti, mentre con il secondo "solamente"

(6000-1)*2 = 11.998.

IMPLEMENTAZIONE IN LINGUAGGIO JAVA (IN LABORATORIO)

Alcune note prima di lavorare ai pc…

- Creeremo un programma "flessibile", che consenta cioè di scegliere quanti elementi inserire ed ordinare. La dimensione del vettore che definiremo dipenderà dall'input dell'utente.
- L'indice usato nel ciclo `for` per confrontare le coppie dovrà arrivare al penultimo elemento del vettore, in modo che si possa confrontare l'elemento `[i]` con l'elemento `[i+1]` senza uscire dal vettore.
- Per lo scambio di valore tra due elementi di un vettore useremo il classico metodo della variabile temporanea.
- Per controllare se il vettore sia ordinato si userà una variabile `boolean` (appunto una *flag*) che verrà messa a `true` all'inizio di ogni ciclo di confronti e diventerà `false` ogni volta che verrà effettuato uno scambio. Il ciclo esterno sarà indeterminato (`do… while`), e continuerà finché la variabile *flag* risulta `false`.
- Per gestire le operazioni da effettuare sul vettore useremo metodi `static` definiti in una classe `Vett`, in modo da poterli richiamare senza bisogno di creare alcun oggetto: **`riempi`** per chiedere in input n elementi interi, **`stampa`** per mettere in output il vettore, **`bubble`** per applicare l'algoritmo di ordinamento. **Avremo così una specie di "officina" in cui il programma `main` invierà il vettore quando avrà bisogno di una operazione su di esso. Questa è l'idea di base dei metodi `static`.**
- E' da sottolineare il fatto che i tre metodi che agiscono sul vettore saranno tutti senza alcun valore di ritorno (`void`); infatti riceveranno come parametro il riferimento al vettore, e lo potranno modificare anche attraverso il parametro stesso.
- Nel metodo `sort` useremo una variabile intera `confr` in cui memorizzeremo il numero di confronti effettuati e un'altra variabile intera `scambi` con la quale conteremo il numero di scambi effettuati per poter apprezzarne la differenza a seconda dello stato iniziale del vettore.

Agli studenti verrà fornito esclusivamente il codice del metodo `main` e la struttura della classe `Vett`, in modo da lavorare su una base comune. I singoli gruppi si cimenteranno nella scrittura dei tre metodi `static` e dei commenti opportuni per la documentazione. Solo in caso di difficoltà verranno aiutati dall'insegnante un passo alla volta.

CODICE FORNITO AGLI STUDENTI

```
import javax.swing.*;

public class Vett
{
    public static void riempi(int[] v, int n)
    {
    }

    public static void stampa(int[] v, int n)
    {
    }

    public static void bubble(int[] v, int n)
    {
    }
}

public class Ordinamento
{
    public static void main(String[] args)
    {
        int n;
        int[] vettore;
        do  n=Integer.parseInt(JOptionPane.showInputDialog("Numero di elementi
(minimo 2)?"));
        while (n<2);
        vettore = new int[n];
        Vett.riempi(vettore,n);
        Vett.stampa(vettore,n);
        JOptionPane.showMessageDialog(null,"Premi OK per applicare l'algoritmo
bubble sort.");
        Vett.bubble(vettore,n);
        Vett.stampa(vettore,n);
    }
}
```

METODO “riempi”

```
/**
 * Metodo per l'inserimento di n elementi interi nel vettore
 *
 * @param v: vettore da riempire
 * @param n: numero di elementi da inserire
 * @return v: vettore riempito
 */
 public static void riempi(int[] v, int n)
    {
        for (int i=0;i<n;i++)
            v[i]=Integer.parseInt(JOptionPane.showInputDialog("Elemento "+i+":"));
    }
```

METODO “stampa”

```
/**
 * Metodo per la stampa a video di n elementi del vettore
 *
 * @param v: vettore da stampare
 * @param n: numero di elementi da stampare
 */
 public static void stampa(int[] v, int n)
    {
        String out="Stato del vettore:\n";
        for (int i=0;i<n;i++)
            out=out+" ["+v[i]+"] ";
        JOptionPane.showMessageDialog(null,out);
    }
```

METODO “bubble”

```
/**
 * Metodo per l'ordinamento del vettore
 *
 * @param v: vettore da ordinare
 * @param n: numero di elementi da ordinare
 * @return v: vettore ordinato
 */
 public static void bubble(int[] v, int n)
    {
        int i,temp,confr,scambi;
        boolean ordinato;
        confr=0;
        scambi=0;
        do
        {
            ordinato=true;
            for(i=0;i<n-1;i++)
            {
                confr++;
                if (v[i]>v[i+1])
                {
                    scambi++;
                    ordinato=false;
                    temp=v[i];      // scambio
                    v[i]=v[i+1];    // di due
                    v[i+1]=temp;    // elementi
                }
            }
         }
         while (!ordinato);
         JOptionPane.showMessageDialog(null,"Il vettore è ordinato.\nHo effettuato
"+confr+" confronti e "+scambi+" scambi.");
    }
}
```