Vettori

Unità didattica 5

Armando Tacchella

Fondamenti di Informatica

Introduzione all'uso dei vettori

- Come gestire 100 oggetti della classe ContoCorrente per mantenere l'insieme dei conti, ad es., aperti in una certa filiale? Utilizzando 100 variabili diverse?
- Come memorizzare le temperature giornaliere sull'arco di un anno per un'indagine meteorologica? Utilizzando 365 variabili diverse?
- La soluzione ad entrambe i problemi è l'uso di un vettore
- Un vettore è una sequenza di dati dello stesso tipo
- Se un programma deve gestire N oggetti dello stesso tipo ($N \geq 2$) probabilmente necessita di un vettore

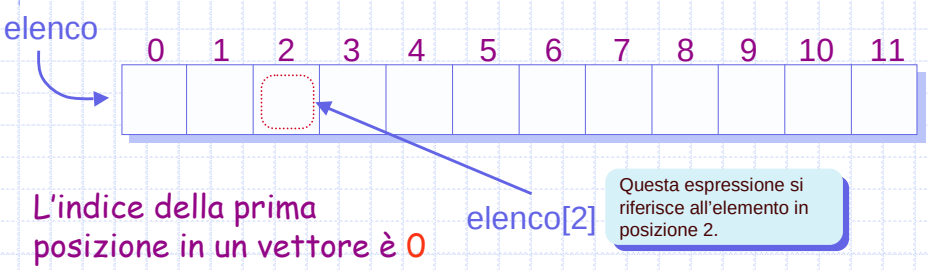
Vettori: dichiarazione e creazione

- I vettori si dichiarano con l'istruzione
  `<tipo di dato>[] <nome>`
  ad esempio:
  ```
  ContoCorrente[] contiFiliale;
  double[] temperature;
  ```
- I vettori si creano con l'istruzione
  `<nome> = new <tipo di dato>[<dimensione>]`
  ad esempio:
  ```
  contiFiliale = new ContoCorrente[100];
  temperature = new float[365];
  ```
- I vettori sono una classe predefinita di Java, pertanto ogni vettore è un oggetto

Vettori: accesso agli elementi (1)

- I singoli elementi di un vettore vengono individuati tramite un indice numerico

```
double[] elenco = new double[12];
```



L'indice della prima posizione in un vettore è 0

Vettori: accesso agli elementi (2)

- La sintassi `<nome>[posizione]` serve sia per leggere sia per impostare un valore in una posizione
- Ad esempio:

```
double[] elenco = new double[12];
elenco[3] = 20.7;
uscitaDati.print("Il valore alla pos. 3 è " +
                 elenco[3]);
```

- La costante pubblica length consente di ottenere la lunghezza del vettore

```
uscitaDati.print("Il vettore consta di " +
                  elenco.lenght + "elementi");
```

- Dato un vettore v, posso accedere a `v[p]` con `0 <= p < v.length`

Vettori: esempi di utilizzo (1)

Il seguente esempio

- dichiara e crea un vettore elenco di 12 elementi
- richiede 12 valori in ingresso all'utente e li memorizza
- calcola la media dei valori inseriti

```
double[] elenco = new double[12];
double media, somma = 0.0;

for (int i = 0; i < elenco.length; i++) {
    elenco[i] = inputBox.getDouble("Valore " + i);
    somma += elenco[i];
}

media = somma / elenco.length;
```

La costante pubblica length restituisce la lunghezza del vettore.

Vettori: esempi di utilizzo (2)

```
double[] voti = new double[4];
String[] materie = new String[4];
materie[0] = “Italiano”;
materie[1] = “Matematica”;
materie[2] = “Inglese”;
materie[3] = “Informatica”;
double media, somma = 0.0;

for (int i = 0; i < voti.length; i++) {
    voti[i] =
      inputBox.getDouble(“Voto di "+ materie[i]);
    somma += voti[i];
}

media = somma / voti.length;
```

Materie relative agli elementi del vettore contenente i voti

Il nome della materia al posto del numero.

Vettori: inizializzazione

- Come altri tipi di dato, è possibile contemporaneamente dichiarare e inizializzare un vettore
- L’inizializzazione ha la seguente sintassi
  `<tipo di dato>[] <nome> = { <elem1>, ..., <elemN> }`
- La dimensione del vettore è pari ad N

```
int[] numeri = {2, 4, 6, 8};

double[] dati = {2.443, 8.99, 12.3, 45.009, 18.2};

String[] materie =
  {“Italiano", “Matematica”, “Inglese", “Informatica”};
```

numeri.length → 4
dati.length → 5
materie.length → 4

Vettori: esempi di utilizzo (3)

```
double[] voti = new double[4];
String[] materie =
  {“Italiano”, “Matematica”, “Inglese”, “Informatica”};
double media, somma = 0.0;

for (int i = 0; i < voti.length; i++) {
    voti[i] =
      inputBox.getDouble(“Voto di "+ materie[i]);
    somma += voti[i];
}

media = somma / voti.length;
```

Inizializzazione del vettore con il nome delle materie

Gestione vettori: introduzione

- Java non mette a disposizione alcun metodo per effettuare le seguenti operazioni sui vettori:
  - inserire di un elemento
  - cercare un elemento
  - cancellare di un elemento
  - confrontare due vettori
  - ordinare due vettori
- Nel seguito realizzeremo una classe VettoreDiInteri che ha lo scopo di illustrare come queste operazioni possano essere realizzate tramite opportuni metodi
- Le tecniche analizzate sono generalizzabili a vettori di qualsiasi tipo

Gestione vettori: definizione di VettoreDiInteri

```
class VettoreDiInteri {

    private int[] vettore;        // vettore
    private int   numeroElementi; // numero di elementi

    public VettoreDiInteri( int numeroMaxElementi ) {
        // Il vettore ha una data lunghezza (massima)
        vettore = new int[numeroMaxElementi];
        // Inizialmente il vettore è vuoto
        numeroElementi = 0;
    }

    // Altri metodi della classe VettoreDiInteri

}
```

Gestione vettori: metodi ausiliari

```
// Metodo per ottenere il numero di elementi del vettore
public int restituisciNumeroElementi( ) {
    return numeroElementi;
}
public int restituisciLunghezza( ) {
    return vettore.length;
}

// Metodo per ottenere il valore di una posizione
public int restituisciElemento( int posizione ) {
    return vettore[posizione];
}

// Metodo per impostare il valore di una posizione
public void impostaElemento( int elemento, int posizione ) {
    vettore[posizione] = elemento;
}
```

Gestione vettori: inserire elementi

- Il metodo funziona secondo la seguente logica:
  - restituisce false se sto tentando di inserire più elementi di quelli che il vettore può contenere (vettore.length)
  - restituisce true se l'elemento può essere inserito
  - l'inserimento avviene in coda nella prima posizione libera
  - `numeroElementi` viene incrementato di 1 in caso di successo

```
public boolean inserisci( int x ) {
    if (numeroElementi < vettore.length) {
        vettore[numeroElementi] = x;
        ++numeroElementi;
        return true;
    } else {
        return false;
    }
}
```

Gestione vettori: cercare elementi

- Il metodo procede come segue
  - Scorre uno ad uno tutti gli elementi del vettore
  - Se l'elemento cercato corrisponde ad uno presente nel vettore restituisce la posizione di quest'ultimo
  - Se nessun elemento corrisponde a quello cercato, restituisce un intero negativo (-1 nel caso specifico)

```
public int cerca( int x ) {
    for (int i = 0; i < vettore.length; i++) {
        if (vettore[i] == x) {
                return i;
        }
    }
    return -1;
}
```

Gestione vettori: cancellare elementi

- Il metodo realizza le seguenti operazioni
  - cerca la posizione dell'elemento da cancellare: se l'elemento non è nel vettore restituisce false; altrimenti, cancella l'elemento e restituisce true
  - la cancellazione avviene in maniera da lasciare il vettore compatto (l'elemento da cancellare è sovrascritto dall'ultimo)

```
public boolean cancella( int x ) {
    int posizione = cerca(x);
    if (posizione >= 0) {
        vettore[posizione] = vettore[numeroElementi – 1];
        --numeroElementi;
        return true;
    } else {
        return false;
    }
}
```

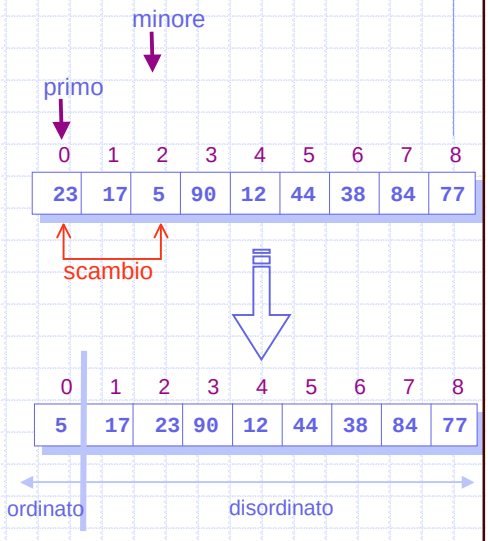
Gestione vettori: confronto di due vettori

- Due vettori u e v sono considerati uguali solo se
  - contengono lo stesso numero di elementi
  - per ogni valore di i, u[i] coincide con v[i]
- Il confronto dei vettori richiede quindi una scansione

```
public boolean confronta(VettoreDiInteri u) {
    if (vettore.length != u.restituisciLunghezza()) {
        return false;
    }
    for (i = 0; i < vettore.length; i++) {
        if (u.restituisciElemento(i) != vettore[i]) {
                return false;
        }
    }
    return true;
}
```

Gestione vettori: ordinamento per selezione (1)

1. Trovare il più piccolo elemento nel vettore
2. Scambiare l'elemento nella prima posizione con quello più piccolo. Adesso l'elemento più piccolo è in prima posizione
3. Ripetere il passo 1 e 2 scartando l'elemento più a sinistra ad ogni passata (l'elemento più piccolo è accantonato dopo ogni passata).



Risultato di una passata

Gestione vettori: ordinamento per selezione (2)

Passata	0	1	2	3	4	5	6	7	8
1	5	17	23	90	12	44	38	84	77
2	5	12	23	90	17	44	38	84	77
3	5	12	17	90	23	44	38	84	77
...									
7	5	12	17	23	38	44	77	84	90
8	5	12	17	23	38	44	77	84	90

ordinato

Risultato DOPO la prima passata

Gestione vettori: ordinamento per selezione (3)

```
public void ordina() {
    for (int i = 0; i < numeroElementi - 1; i++) {
        // Ricerca del minimo
        int posizioneMin = i;
        for (int j = i + 1; i < numeroElementi; j++) {
                if (vettore[j] < vettore[posizioneMin]) {
                        posizioneMin = j;
                }
        }
        // Scambio del minimo con il primo
        int t = vettore[i];
        vettore[i] = vettore[posizioneMin];
        vettore[posizioneMin] = t;
    }
}
```

Vettori di oggetti: introduzione

- Combinando i vettori e gli oggetti è possibile creare strutture dati complesse ed efficaci
- La gestione dei vettori di oggetti è analoga a quella dei vettori di tipi primitivi
- Dato che ogni elemento del vettore è un oggetto, è necessario crearlo prima di poterlo utilizzare
- Ad esempio:

```
String[] elenco = new String[100];
elenco[0] = "Java";
int p = elenco[0].indexOf('a');  // OK, p vale 1
int q = elenco[1].indexOf('a');  // Errore!
```

- elenco[1] non corrisponde ad alcun oggetto (vale null)

Vettori di oggetti: la classe Persona (1)

```
class Persona {
    private String nome;
    private int    anni;
    private char   genere;

    public Persona( String n, int a, char g) {
        nome = n;
        anni = a;
        genere = g;
    }
    public Persona( ) {
        nome = "";
        anni = 0;
        genere = '?';
    }
```

Vettori di oggetti: la classe Persona (2)

```
    public String restituisciNome( ) {
        return nome;
    }
    public void impostaNome( String n ) {
        nome = n;
    }
    public int restituisciAnni( ) {
        return anni;
    }
    public void impostaAnni( int a ) {
        anni = a;
    }
    // Metodi di impostazione/restituzione per il genere
}
```

Vettori di oggetti: creazione (1)

Sorgente

A

```
Persona[ ]  elenco;
elenco = new Persona[20];
elenco[0] = new Persona( );
```

Viene dichiarato solo il nome elenco, nessun vettore viene allocato.

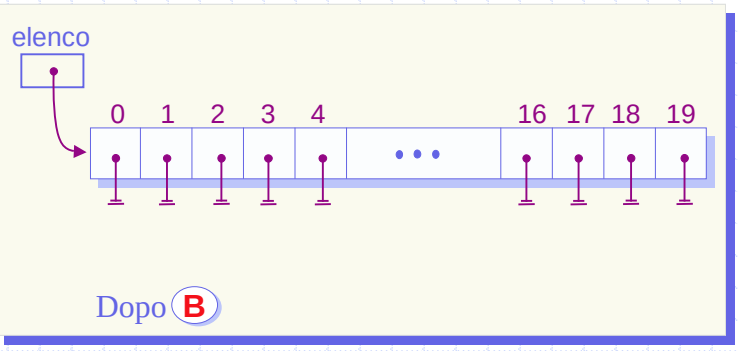
Vettori di oggetti: creazione (2)

Sorgente

B

```
Persona[ ]  elenco;
elenco = new Persona[20];
elenco[0] = new Persona( );
```

Il vettore per contenere 20 oggetti Persona viene creato, ma gli oggetti stessi sono ancora null.



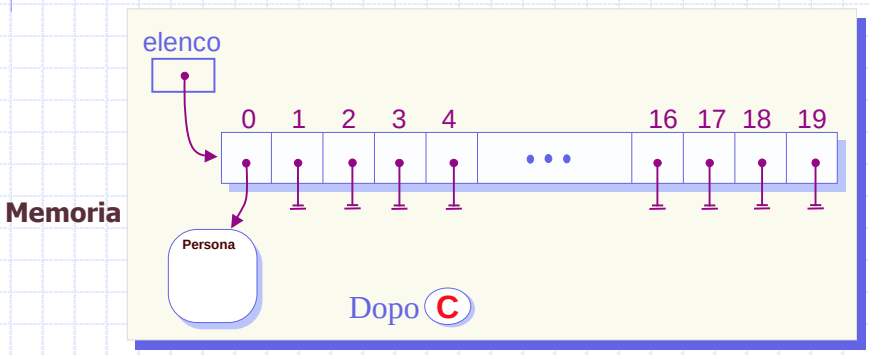
Vettori di oggetti: creazione (3)

Sorgente

```
Persona[ ]  elenco;
elenco = new Persona[20];
elenco[0] = new Persona( );
```

C

Un oggetto Persona viene creato e il riferimento a questo oggetto è assegnato alla posizione 0.



Vettori di oggetti: esempi di utilizzo (1)

```
// Crea un vettore di oggetti Persona
Persona[] elenco = new Persona[20];

// Crea un oggetto persona per ogni elemento
for (int i = 0; i < elenco.length; i++) {
    String nome = inDati.getString("Nome:");
    int    anni = inDati.getInteger("Età:");
    String temp = inDati.getString("Genere:");

    elenco[i] = new Persona(nome, anni, temp.charAt(0));
}
```

Vettori di oggetti: esempi di utilizzo (2)

```
// Calcola l'età media dei maschi con più di 18 anni
double media = 0.0;
int maggiorenni = 0;
for (int i = 0; i < elenco.length; i++) {
    if ((elenco[i].restituisciAnni( ) > 18) &&
        (elenco[i].restituisciGenere( ) == 'M')) {
        media += elenco[i].restituisciAnni();
        ++maggiorenni;
    }
}
if (maggiorenni > 0) {
    media = media / maggiorenni;
}
```

Vettori di oggetti: esempi di utilizzo (3)

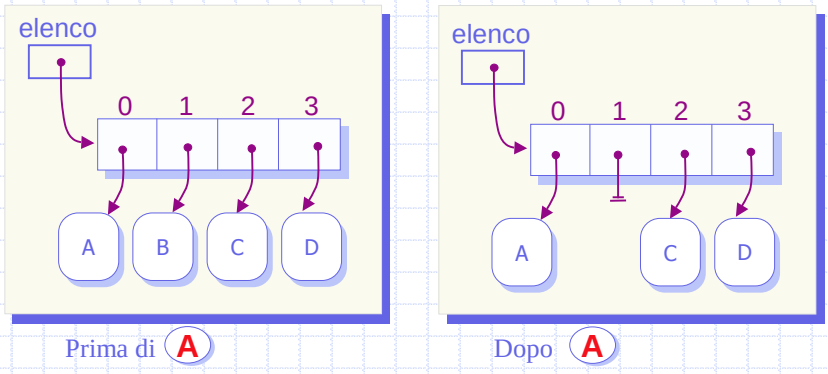
```
// elenco è già stato creato e riempito
// Trova la posizione del più giovane e del più anziano
int minPos = 0, maxPos = 0;
for (int i = 1; i < elenco.length; i++) {
    if ( elenco[i].restituisciAnni() <
         elenco[minPos].restituisciAnni()) {
        minPos = i;
    } else if (elenco[i].restituisciAnni() >
               elenco[maxPos].restituisciAnni()) {
        maxPos = i;
    }
}
// elenco[minPos] è il più giovane
// elenco[maxPos] è il più anziano
```

Vettori di oggetti: cancellazione di elementi (1)

A

```
int posizione = 1;
elenco[posizione] = null
```

Cancella B impostando la posizione 1 al valore null.



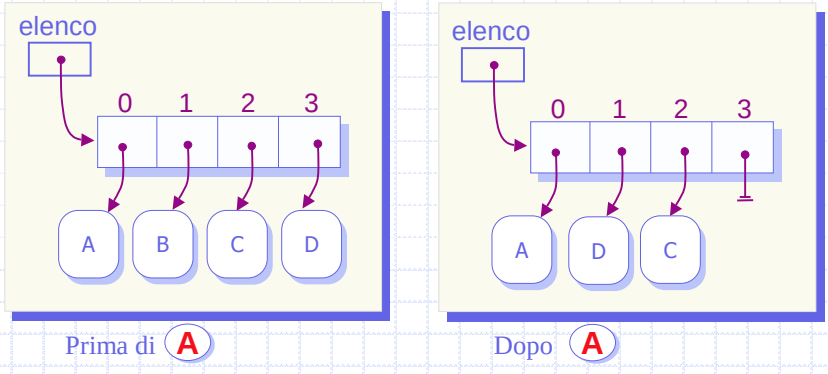
Prima di A Dopo A

Vettori di oggetti: cancellazione di elementi (2)

A

```
int posizione = 1, fine = 3;
elenco[posizione] = elenco[fine];
elenco[fine]      = null;
```

Cancella B impostando la posizione 1 al valore dell'ultima posizione, e l'ultima posizione a null.



Prima di A Dopo A

Matrici (vettori bidimensionali)

- Le matrici sono utili per la rappresentazione di dati che dipendono da due parametri (ad. es. tavola pitagorica).
- La matrice è un vettore avente vettori come elementi
- Le matrici si dichiarano con l'istruzione
  `<tipo di dato>[][] <nome>`
  ad esempio:
  `int[][] tavolaPitagorica;`
- Le matrici si creano con l'istruzione
  `<nome> = new <tipo di dato>[<dim1>][<dim2>]`
  ad esempio:
  `tavolaPitagorica = new int[10][10];`

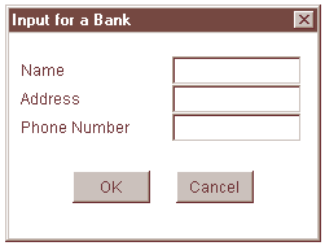
Matrici: esempio di utilizzo

```
// Calcola la media di ogni riga
double[ ] media = new double[tavolaPitagorica.length];

for (int i = 0; i < tavolaPitagorica.length; i++) {

    for (int j = 0; j < tavolaPitagorica[i].length; j++) {

        media[i] += tavolaPitagorica[i][j];
    }

    media[i] = media[i] / tavolaPitagorica[i].length;
}
```

Classe MultiInputBox (javabook)

- Consente di inserire più valori contemporaneamente.

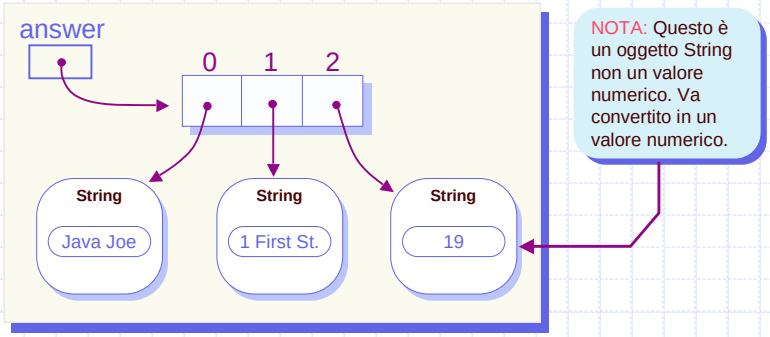
```
String[] label = {"Name", "Address", "Phone Number"};

MultiInputBox mBox = new MultiInputBox( mainWindow, 3 );
mBox.setLabels( label );
mBox.setTitle("Input for a Bank");
String[] answer = mBox.getInputs( );
```



Metodo getInputs

- Il metodo getInputs restituisce un vettore di String.

```
String[] answer = mBox.getInputs( );
```



Metodi di MultiInputBox

Metodo	Argomenti	Descrizione
`MultiInputBox (costruttore)`	`MainWindow, int`	Crea un oggetto MultiInputBox. Il secondo argomento specifica il numero di campi di inserimento testo.
`MultiInputBox (costruttore)`	`MainWindow, vettore di String`	Crea un oggetto MultiInputBox. Il secondo argomento è un vettore di String per le etichette dei campi di inserimento testo.
`setLabels`	`vettore di String`	Imposta le etichette dei campi sulla base degli elementi del vettore passato.
`getInputs`	`<nessuno>`	Restituisce un vettore di String con i dati inseriti dall'utente nei campi di testo.

Esercitazione – Vettori

- Aggiungere i seguenti metodi alla classe VettoreDiInteri:
  - somma ad un vettore delle stesse dimensioni
  - prodotto di tutti gli elementi per una costante k
  - prodotto scalare con un altro vettore
  - somma e media di tutti gli elementi
  - ricerca della posizione del minimo e del massimo elemento
- Definire una classe VettoreDiCaratteri che includa i seguenti metodi
  - rivelatore di parole palindrome (ad es. ara, otto, e idi)
  - conversione in un VettoreDiInteri corrispondente ai codici dei caratteri
  - conversione da un VettoreDiInteri contenente i codici dei caratteri

Esercitazione – Vettori e classi

- Definire una classe Persona con i seguenti attributi:
  - Nome (nome e cognome della persona)
  - Indirizzo (via, strada o piazza e numero civico)
  - Telefono (incluso il prefisso)
- Definire una classe Rubrica che consenta di:
  - Aggiungere una persona
  - Cercare una persona per nome
  - Cercare tutte le persone che abitano ad un dato indirizzo
  - Cancellare una persona (dato il nome)
- Prevedere l'aggiunta di persone oltre il limite iniziale fissato per la Rubrica (suggerimento: quando il vettore è pieno, se ne crea uno più ampio e si copiano gli elementi)

Esercitazione – Vettori e classi

- Definire una classe per la gestione di tempi e orari espressi in ore e minuti
- Definire una classe Appuntamento che contenga:
  - Ora dell'appuntamento
  - Durata dell'appuntamento
  - Nome della persona da incontrare
  - Descrizione
- Definire una classe AgendaGiornaliera che consenta di:
  - inserire un appuntamento (controllando le sovrapposizioni)
  - cancellare un appuntamento
  - cercare un appuntamento per nome
  - cercare tutti gli appuntamenti da una certa ora in poi

Unità didattica 5 – Argomenti svolti

- Dichiarazione, creazione, inizializzazione di vettori
- Manipolazione di vettori: inserimento, cancellazione, ricerca di elementi; confronto e ordinamento di vettori
- Differenza tra vettori di tipi primitivi e vettori di oggetti
- Matrici (vettori bidimensionali)
- Classe MultiInputBox del package javabook