Lezione 8

Vettori

Lezione 8 - Obiettivi

Al termine di questa lezione saremo in grado di

- Manipolare sequenze di valori utilizzando vettori.
- Dichiarare e utilizzare vettori di tipi primitivi.
- Dichiarare e utilizzare vettori di oggetti.
- Descrivere la realizzazione di una matrice come vettore di vettori.
- Manipolare sequenze di dati utilizzando i vettori.
- Utilizzare l'oggetto MultiInputBox del package javabook per inserire vettori di stringhe.
- Definire metodi che accettano un vettore come parametro e metodi che ritornano vettori.
- Descrivere e utilizzare il puntatore di auto-riferimento.

Vettori: Elementi Fondamentali

- Si assuma di dover gestire 100 oggetti della classe Student per mantenere la lista degli studenti iscritti ad un dato corso. E' pensabile farlo con 100 variabili diverse?
- Si assuma di dover conservare le temperature giornaliere su un arco di dodici mesi in un programma di indagine meteorologica. E' pensabile realizzare l'elenco con 365 variabili diverse?
- Naturalmente è pensabile, ma non è pratico!
- Un vettore è una sequenza di dati dello stesso tipo.
- Se un programma deve gestire N oggetti dello stesso tipo (N ≥ 2) probabilmente necessita di un vettore.

Vettori di Tipi Primitivi

- Dichiarazione di vettori

```
<tipo di dato> [ ] <variabile>  // variante 1
<tipo di dato>   <variabile>[ ] // variante 2
```

- Creazione di vettori

```
<variabile>  = new <tipo di dato> [ <dimensione> ]
```

- Esempi

Variante 1

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

Variante 2

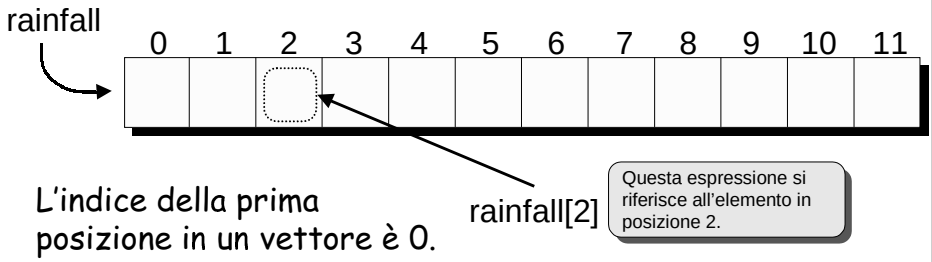
```
double rainfall [ ];
rainfall
   = new double[12];
```

Un vettore è un oggetto!

Accesso ai Singoli Elementi

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

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



L'indice della prima posizione in un vettore è 0.

Utilizzo di Vettori – 1

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

double   annualAverage,
         sum = 0.0;

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

    rainfall[i] = inputBox.getDouble("Piogge del mese "
                                     + (i+1) );
    sum += rainfall[i];
}

annualAverage = sum / rainfall.length;
```

La costante pubblica length restituisce la lunghezza del vettore.

Utilizzo di Vettori – 2

```
double[] rainfall = new double[12];
String[] monthName = new String[12];
monthName[0] = “Gennaio”;
monthName[1] = “Febbraio”;
…

double   annualAverage, sum = 0.0;

for (int i = 0; i < rainfall.length; i++) {
    rainfall[i] = inputBox.getDouble(“Piogge nel mese di "
                                     + monthName[i] );
    sum += rainfall[i];
}
annualAverage = sum / rainfall.length;
```

Stesso schema per i restanti mesi.

Il nome del mese al posto del numero.

Utilizzo di Vettori – 3

- Calcolo delle precipitazioni medie per trimestre.

```
//rainfall è dichiarato e inizializzato opportunamente

double[] quarterAverage = new double[4];

for (int i = 0; i < 4; i++) {
    sum = 0;
    for (int j = 0; j < 3; j++) {
                                        // calcola la somma
        sum += rainfall[3*i + j];       // di un trimestre
    }
    quarterAverage[i] = sum / 3.0;   // media del trimestre
}
```

Inizializzazione di Vettori

- Come altri tipi di dato, è possibile contemporaneamente dichiarare e inizializzare un vettore.

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

double[] samplingData = { 2.443, 8.99, 12.3, 45.009, 18.2,
                          9.00, 3.123, 22.084, 18.08 };

String[] monthName = { “Gennaio", "Febbraio", "Marzo",
                       "Aprile", "Maggio", “Giugno", “Luglio",
                       "Agosto", "Settembre", "Ottobre",
                       "Novembre", "Decembre"  };
```

- La dimensione del vettore è pari al numero di elementi nella lista.

```
number.length ⟶ 4
samplingData.length ⟶ 9
monthName.length ⟶ 12
```


Vettori di Oggetti

- Combinando i vettori e gli oggetti è possibile creare strutture dati complesse ed efficaci.
- Avendo a disposizione solo vettori su tipi primitivi, la rappresentazione di una sequenza di oggetti richiederebbe l’utilizzo di diversi vettori, uno per ogni attributo degli oggetti. Questa tecnica è chiaramente macchinosa e soggetta ad errori.
- Un vettore di oggetti consente una rappresentazione più concisa e risulta in un codice sorgente più comprensibile ed elegante.

La Classe Person

- Un esempio di classe per illustrare l'uso dei vettori di oggetti.

```
Person latte;

latte = new Person( );
latte.setName("Ms. Latte");
latte.setAge(20);
latte.setGender('F');

outputBox.printLine( "Nome  : " + latte.getName()   );
outputBox.printLine( "Età   : " + latte.getAge()    );
outputBox.printLine( "Sesso : " + latte.getGender() );
```

La classe Person supporta i metodi set e get per ogni suo attributo.

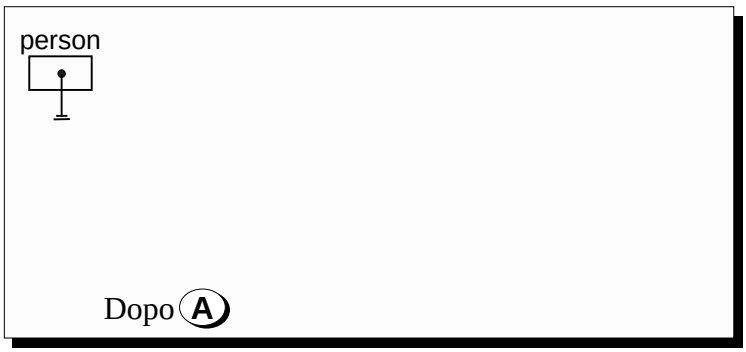
Creazione di un Vettore di Oggetti - 1

Sorgente (A)

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

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

Memoria



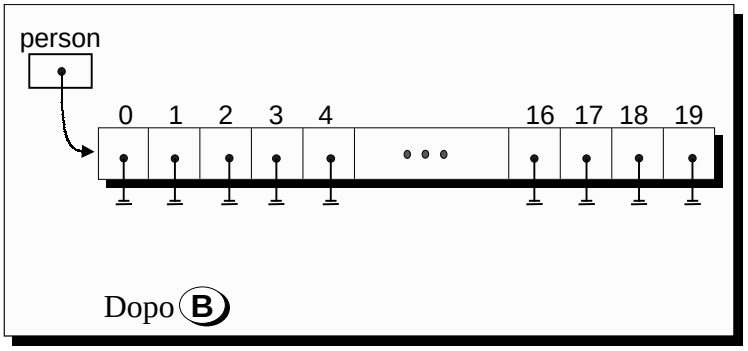
Creazione di un Vettore di Oggetti - 2

Sorgente

B

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

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



 Introduction to Object-Oriented Programming with Java--Wu Chapter 9 - 13

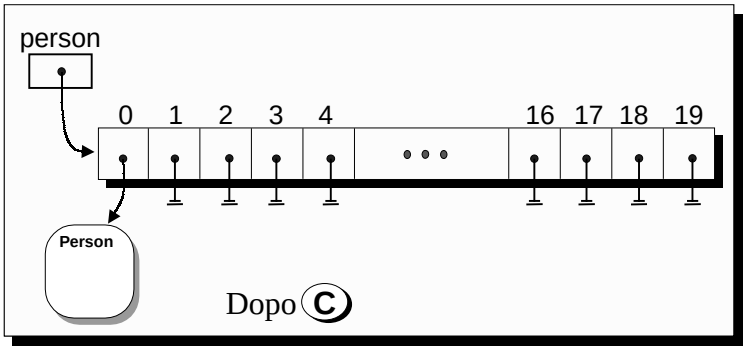
Creazione di un Vettore di Oggetti - 3

Sorgente

C

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

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



 Introduction to Object-Oriented Programming with Java--Wu Chapter 9 - 14

Gestione di un Vettore di Oggetti - 1

- Creazione degli oggetti Person e memorizzazione nel vettore person.

```
String      name, inpStr;
int         age;
char        gender;

for (int i = 0; i < person.length; i++) {
    name   = inputBox.getString("Nome:"); //legge i valori degli attributi
    age    = inputBox.getInteger("Età:");
    inpStr = inputBox.getString("Sesso:");
    gender = inpStr.charAt(0);

    person[i] = new Person( );            //crea un oggetto Person

    person[i].setName  ( name   );        //imposta gli attributi
    person[i].setAge   (  age   );
    person[i].setGender( gender );
}
```


Gestione di un Vettore di Oggetti - 2

- Trova l'età media dei maschi con più di 18 anni.

```
float sum = 0, averageAge;
int   count = 0;

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

    if ( person[i].getAge( ) > 18 &&
         person[i].getGender{ } = 'M' ) {
        sum += person[i].getAge();
    }
}

averageAge = sum / count;
```


Gestione di un Vettore - 3

- Trova il più giovane e il più anziano.

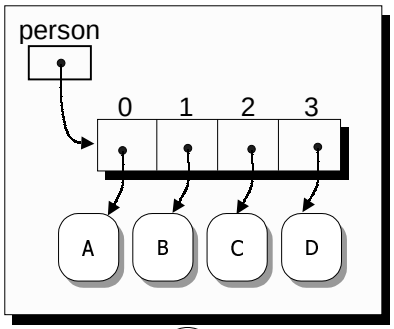
```
int     minIdx = 0;      //indice al più giovane
int     maxIdx = 0;      //indice al più anziano

for (int i = 1; i < person.length; i++) {

    if ( person[i].getAge() < person[minIdx].getAge() ) {
        minIdx       = i;         //trovata una persona più giovane
    }
    else if (person[i].getAge() > person[maxIdx.getAge() ) {

        maxIdx        = i;        //trovata una persona più anziana
    }
}
//person[minIdx] è il più giovane e person[maxIdx] è il più anziano
```


Cancellazione di Oggetti – Approccio 1

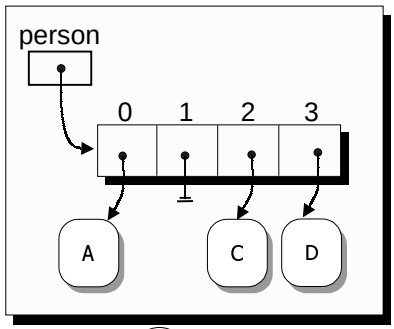
A

```
int delIdx = 1;
person[delIdx] = null
```

Cancella B impostando la posizione 1 al valore null.



Prima di A



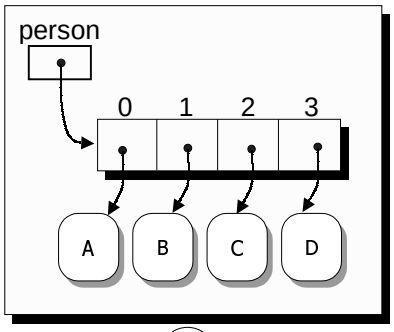
Dopo A

Cancellazione di Oggetti – Approccio 2

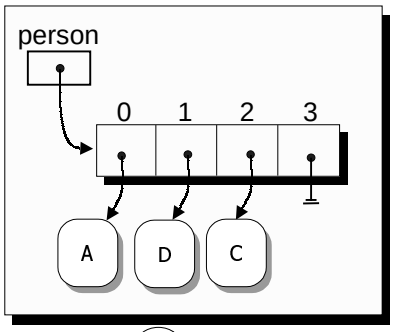
A

```
int delIdx = 1, last = 3;
person[delIndex] = person[last];
person[last]     = null;
```

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



Prima di A

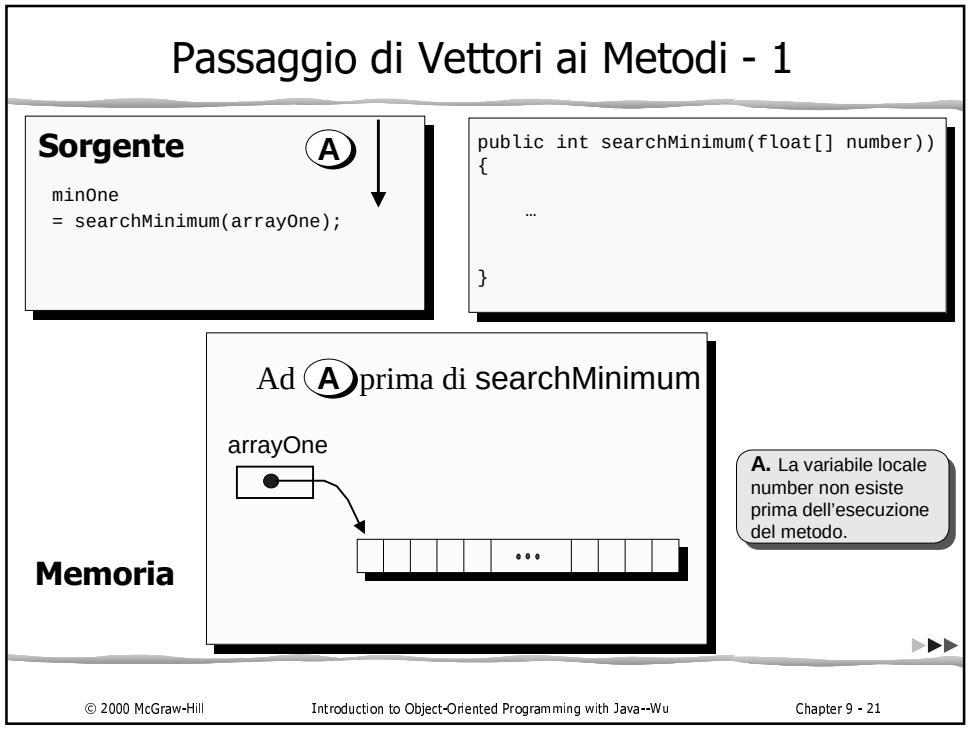


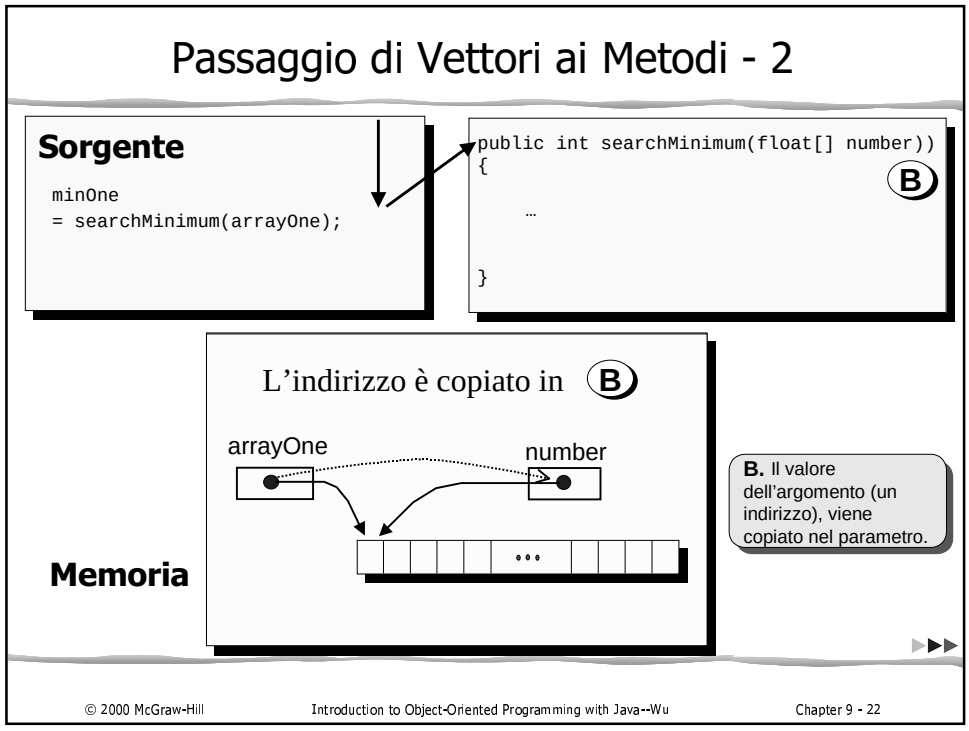
Dopo A

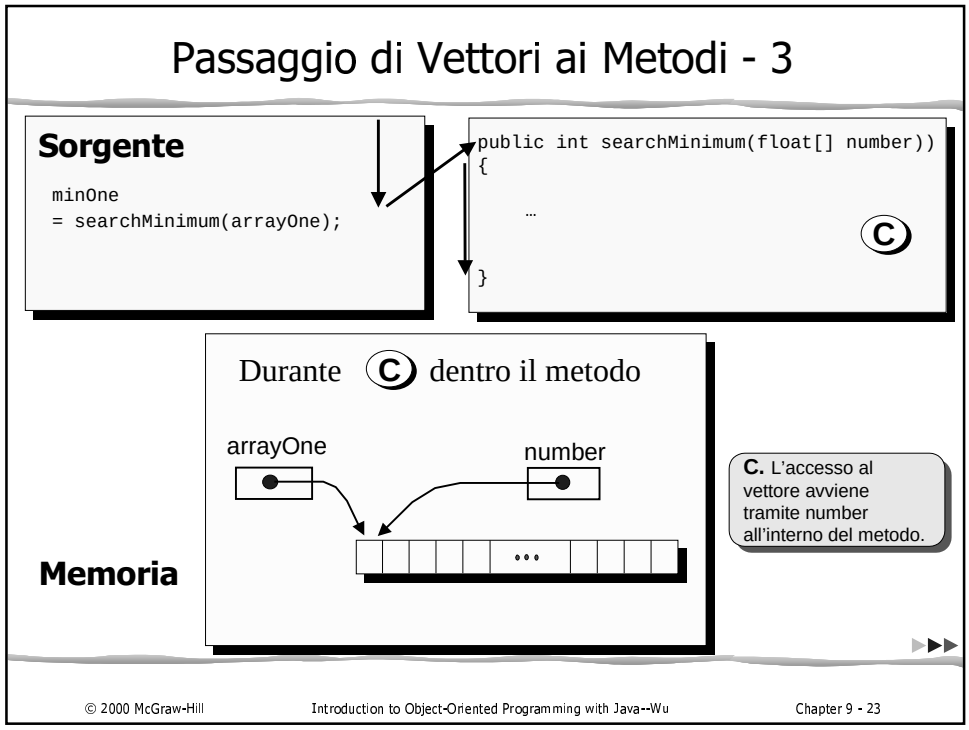
Gestione di un Vettore di Oggetti - 4

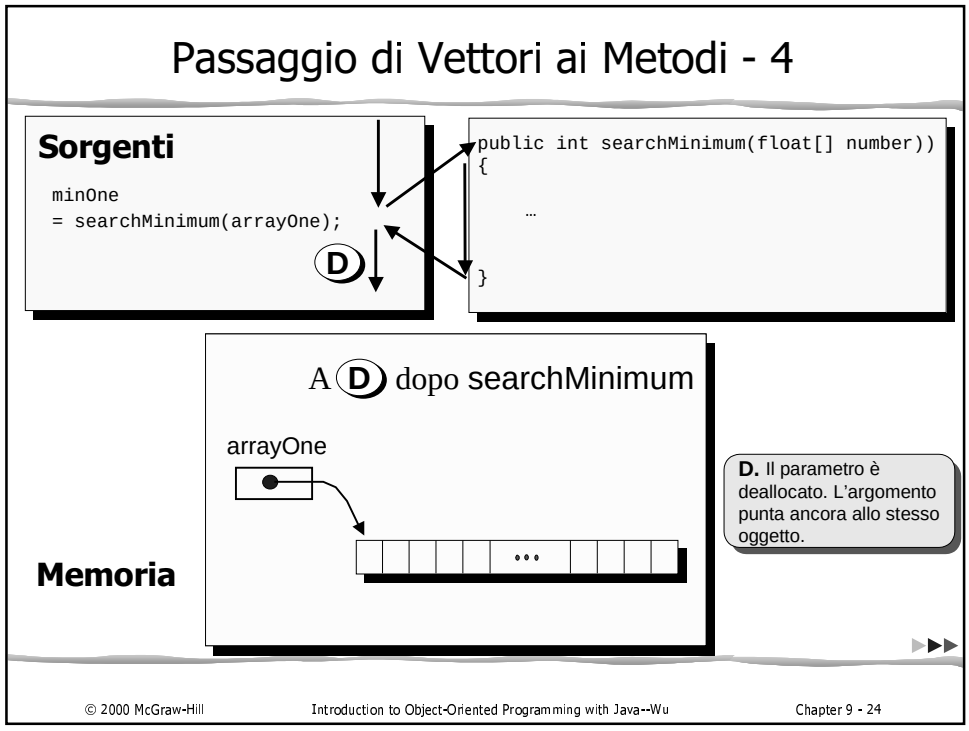
- Ricerca di una persona.

```
int i = 0;

while ( person[i] != null && !person[i].getName().equals("Latte") ) {
    i++;
}

if ( person[i] == null ) {
    //non trovato – ricerca senza successo
    outputBox.printLine("Ms. Latte non è nel vettore");
}
else {
    //trovato – ricerca con successo
    outputBox.printLine("Ho trovato Ms. Latte alla posizione " + i);
}
```



Passaggio di Vettori ai Metodi - 1
Sorgente
A
minOne
= searchMinimum(arrayOne);
public int searchMinimum(float[] number))
{
…
}
Ad A prima di searchMinimum
arrayOne
Memoria
A. La variabile locale number non esiste prima dell'esecuzione del metodo.
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 9 - 21


Passaggio di Vettori ai Metodi - 2
Sorgente
minOne
= searchMinimum(arrayOne);
public int searchMinimum(float[] number))
{
B
…
}
L'indirizzo è copiato in B
arrayOne
number
Memoria
B. Il valore dell'argomento (un indirizzo), viene copiato nel parametro.
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 9 - 22

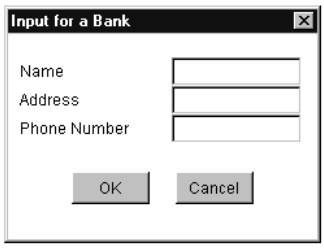

Passaggio di Vettori ai Metodi - 3
Sorgente
minOne
= searchMinimum(arrayOne);
public int searchMinimum(float[] number))
{
…
}
C
Durante C dentro il metodo
arrayOne
number
Memoria
C. L'accesso al vettore avviene tramite number all'interno del metodo.
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 9 - 23


Passaggio di Vettori ai Metodi - 4
Sorgenti
minOne
= searchMinimum(arrayOne);
D
public int searchMinimum(float[] number))
{
…
}
A D dopo searchMinimum
arrayOne
Memoria
D. Il parametro è deallocato. L'argomento punta ancora allo stesso oggetto.
© 2000 McGraw-Hill
Introduction to Object-Oriented Programming with Java--Wu
Chapter 9 - 24

MultiInputBox

- La classe MultiInputBox è utilizzata per inserire più valori contemporaneamente.

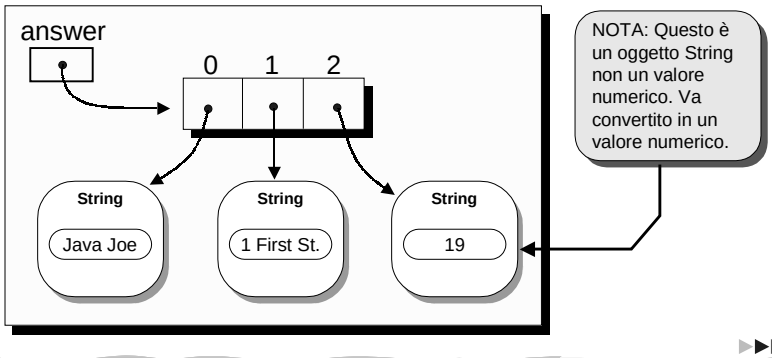
```
String[] label = {"Name", "Address", "Phone Number"};

MultiInputBox multiBox = new MultiInputBox( mainWindow, 3 );
multiBox.setLabels( label );
multiBox.setTitle("Input for a Bank");

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



MultiInputBox - getInputs

- Il metodo getInputs restituisce un vettore di oggetti String.

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



MultiInputBox Metodi

CLASS:	MultiInputBox	
Method	Argument	Description
<constructor>	MainWindow, int	Creates a MultiInputBox object. The second argument specifies the number of labels.
<constructor>	MainWindow, array of String	Creates a MultiInputBox object. The second argument is an array of String for labels.
setLabels	array of String	Sets the labels of a MultiInputBox object to the passed array of String.
getInputs	<none>	Returns an array of String entered by the user.

Puntatore di auto-riferimento

- La parola chiave this è utilizzata per riferirsi esplicitamente agli attributi e ai metodi di un oggetto (nonchè all'oggetto stesso).
- Sino ad ora, la parola chiave this non è stata utilizzata in quando non si verificavano ambiguità.

```
class Tester
{
    public void m1( )
    {
    }

    public void m2( )
    {
        m1( );
    }
}
```

```
class Tester
{
    public void m1( )
    {
    }

    public void m2( )
    {
        this.m1( );
    }
}
```


Variabili Locali vs. Attributi

- A cosa si riferisce l'identificatore age?

```
class Person
{
  int age;
  ...
  public void setAge( int age )
  {
     ... age ...

  }
}
```

C'è un corrispondente parametro locale e quindi age si riferisce al parametro.

```
class Person
{
    int age;
    ...
    public void setAge( int pAge )
    {
       ... age ...

    }
}
```

Non c'è un corrispondente parametro locale e quindi age si riferisce all'attributo.

Come Evitare i Conflitti con this

```
class Person
{
  int age;
  ...
  public void setAge( int age )
  {
    this.age = age;

  }
}
```

Chiamata di un Costruttore con this

- La parola chiave this può essere utilizzata per chiamare un costruttore all'interno di un altro costruttore della stessa classe.

```
class Person
{
  public void Person( )
  {
     this( "Not Given", 0, 'U' );
  }
  public void Person( String name, int age, char gender )
  {
     this.name = name;
     this.age  = age;
     this.gender = gender;
  }
     …
}
```

Questa classe ha due costruttori. Il primo costruttore chiama il secondo con valori di default per gli argomenti.

Esempio di Sviluppo: AddressBook

- Problema

  Scrivere una classe AddressBook *che gestisca un elenco di persone. Un oggetto* AddressBook *deve consentire al programmatore di aggiungere, cancellare e cercare una persona dell'elenco.*

- Schema di massima

Metodo	Descrizione
`<constructor>`	Un costruttore per inizializzare l'oggetto. Costruttori multipli possono essere previsti se necessario.
`add`	Aggiunge un nuovo oggetto Person all'elenco.
`delete`	Cancella uno specifico oggetto Person dall'elenco.
`search`	Ricerca uno specifico oggetto Person nell'elenco e lo restituisce qualora lo trovi.

AddressBook – Passi di Sviluppo

1. Implementare il(i) costruttore(i).
2. Implementare il metodo add.
3. Implementare il metodo search.
4. Implementare il metodo delete.
5. Finalizzare la classe.

Vettori Bi-Dimensionali

- I vettori Bi-Dimensionali sono utili per la rappresentazione di dati sotto forma di matrici.

Distance Table (in miles)

	Los Angeles	San Francisco	San Jose	San Diego	Monterey
Los Angeles	—	600	500	150	450
San Francisco	600	—	100	750	150
San Jose	500	100	—	650	50
San Diego	150	750	650	—	600
Monterey	450	150	50	600	—

Multiplication Table

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

Tuition Table

	Day Students	Boarding Students
Grades 1 – 6	$ 6,000.00	$ 18,000.00
Grades 7 – 8	$ 9,000.00	$ 21,000.00
Grades 9 – 12	$ 12,500.00	$ 24,500.00

Dichiarazione e Creazione di Vettori 2-D

- Dichiarazione

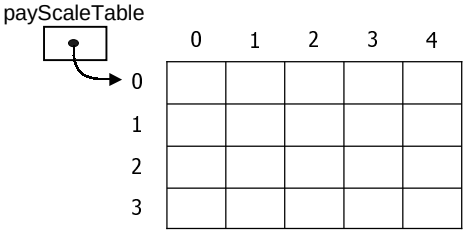
```
<tipo di dato> [][] <variabile> //variante 1
<tipo di dato>   <variable>[][] //variante 2
```

- Creazione

```
<variabile>  = new <tipo di dato> [ <dim1> ][ <dim2> ]
```

- Esempio

```
double[][] payScaleTable;
payScaleTable
    = new double[4][5];
```



Esempio di Utilizzo di Vettori 2-D

- Trovare la media di ogni riga.

```
double[ ] average = { 0.0, 0.0, 0.0, 0.0 };

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

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

        average[i] += payScaleTable[i][j];
    }

    average[i] = average[i] / payScaleTable[i].length;
}
```

La Classe Vector

- La classe Vector, definita nel package standard java.util, realizza un vettore di oggetti dinamico.
- Si possono aggiungere nuovi elementi ad un oggetto Vector senza preoccuparsi di eccedere la sua capacità.
- Internamente, la classe Vector mantiene un vettore di oggetti, e quando viene superata la capacità di tale vettore, si utilizza un vettore più grande per rimpiazzarlo.
- Gli elementi di un oggetto Vector devono essere oggetti a loro volta; non possono essere tipi primitivi quali int e double.

Utilizzo di Vector - 1

- Creazione di un vettore dinamico friends e due oggetti Person.

```
import java.util.*;
Vector friends;
Person person;

friends = new Vector( );

person = new Person("jill", 10, 'F');
friends.add( person );

person = new Person("jack",  6, 'M');
friends.add( person );
```

La classe Vector è definita in questo package.

Crea un nuovo vector con una capacità pari a 10.

Aggiunge una nuova Person alla fine.

Utilizzo di Vector - 2

- Visualizza i nomi di tutti gli oggetti Person nel vettore dinamico friends.

```
Person p;

int limit = friends.size( );

for (int i = 0; i < limit; i++ ) {

    p = (Person) friends.elementAt( i );

    System.out.println( p.getName( ) );
}
```

Il metodo size restituisce il numero di oggetti nel vettore.

Il metodo elementAt restituisce l'oggetto nella posizione i.

Utlizzo di Vector - 3

- Per memorizzare dati di tipo primitivo in un Vector è necessario utilizzare le classi "wrapper" (avvolgenti) Integer, Double, etc.

```
Vector intVector = new Vector( );
intVector.add( new Integer( 15 ) );
intVector.add( new Integer( 30 ) );
...
Integer intObject;
Enumeration enum = intVector.elements( );

while ( enum.hasMoreElements( ) ) {
    intObject = (Integer) enum.nextElement( );
    System.out.println( intObject.intValue( ) );
}
```

Il metodo intValue della classe Integer restituisce un valore intero con tipo di dato primitivo int.

The End

