Questo capitolo introduce alla programmazione in Java, in modo superficiale, per dare un'idea delle potenzialità di questo linguaggio.

Struttura fondamentale

«

Java è un linguaggio di programmazione strettamente OO (*Object oriented*), cioè a dire che qualunque cosa si faccia, anche un semplice programma che emette un messaggio attraverso lo standard output, va trattato secondo la programmazione a oggetti.

Ciò significa anche che i componenti di questo linguaggio hanno nomi diversi da quelli consueti. Volendo fare un abbinamento approssimativo con un linguaggio di programmazione normale, si potrebbe dire che in Java i programmi sono ***classi*** e le funzioni sono ***metodi***. Naturalmente ci sono anche tante altre cose nuove.

Fatta questa premessa, si può dare un'occhiata alla solita classe banale: quella che visualizza un messaggio e termina.

```
/**
 *  CiaoMondoApp.java
 *  La solita classe banale.
 */

import java.lang.*; // predefinita

class CiaoMondoApp
{
    public static void main (String[] args)
    {
        System.out.println ("Ciao Mondo!"); // visualizza il messaggio
    }
}
```

Il sorgente Java ha molte somiglianze con quello del linguaggio C e qui si intendono segnalare le particolarità rispetto a quel linguaggio.

Commenti

Java ammette l'uso di commenti in stile C, nella solita forma '**/*...*/**', ma ne introduce altri due tipi: uno per la creazione automatica di documentazione, nella forma '**/**...*/**', e uno per fare ignorare tutto ciò che appare a partire dal simbolo di commento fino alla fine della riga, nella forma '***//commento***':

```
/* commento_generico */
```

```
/** documentazione */
```

```
// commento_fino_alla_fine_della_riga
```

Tutti e tre questi tipi di commenti servono a fare ignorare al compilatore una parte del sorgente e questo dovrebbe bastare al prin-

cipiante. Convenzionalmente, è conveniente usare il commento di documentazione per la spiegazione di ciò che fa la classe, all'inizio del sorgente.

Nomi ed estensioni

«

Le estensioni dei file Java sono definite in modo obbligatorio: '`.java`' per i sorgenti e '`.class`' per le classi (i binari Java).

Generalmente, nel sorgente, il nome della classe deve corrispondere alla radice del nome del sorgente e, di conseguenza, anche del binario Java. Per lo stile convenzionale di Java, questo nome inizia con una lettera maiuscola e non contiene simboli strani; se è composto dall'unione di più parole, ognuna di queste inizia con una lettera maiuscola.

Istruzioni

«

Le istruzioni seguono la convenzione del linguaggio C, per cui terminano con un punto e virgola ('`;`') e i raggruppamenti di queste, detti anche blocchi, si fanno utilizzando le parentesi graffe ('`{ }`').

```
istruzione ;
```

```
{ istruzione ;  istruzione ;  istruzione ;  }
```

Generalmente, un'istruzione può essere interrotta e ripresa nella riga successiva, dal momento che la sua conclusione è dichiarata chiaramente dal punto e virgola finale.

Ogni programma in Java deve fare affidamento sull'utilizzo di classi fondamentali che compongono il linguaggio stesso. L'importazione delle classi necessarie viene fatta attraverso l'istruzione '**`import`**', indicando una classe particolare o un gruppo (nel secondo caso si usa un asterisco).

Nell'esempio introduttivo vengono importate tutte le classi del pacchetto '**`java.lang`**', anche se non sarebbe stato necessario dichiararlo, dato che queste classi vengono sempre importate in modo predefinito (senza di queste, nessuna classe potrebbe funzionare).

Le classi standard di Java (cioè queste librerie fondamentali), sono contenute normalmente in un archivio compresso '`.zip`', oppure '`.jar`'. Si è visto nel capitolo u122 che è importante indicare il percorso in cui si trovano, nella variabile di ambiente '**`CLASSPATH`**'.

Osservando il contenuto di questo file, si può comprendere meglio il concetto di pacchetto di classi. Segue solo un breve estratto:

```
Archive:  classes.zip
 Length     Date    Time    Name
 ------     ----    ----    ----
      0  05-19-97  22:46   java/
      0  05-19-97  22:24   java/lang/
   1322  05-19-97  22:24   java/lang/Object.class
   4202  05-19-97  22:24   java/lang/Class.class
    …
   3450  05-19-97  22:24   java/lang/System.class
    …
      0  05-19-97  22:26   java/util/
    …
      0  05-19-97  22:26   java/io/
    …
      0  05-19-97  22:42   java/awt/
    …
```

Ecco che così può diventare più chiaro il fatto che, importare tutte le classi del pacchetto '**`java.lang`**' significa in pratica includere tutte le classi contenute nella directory '`java/lang/`', anche se qui si tratta solo di un file compresso.

Dichiarazione della classe

Generalmente, un file sorgente Java contiene la dichiarazione di una sola classe, il cui nome corrisponde alla radice del file sorgente. La dichiarazione della classe delimita in pratica il contenuto del sorgente, definendo eventuali ***ereditarietà*** da altre classi esistenti.

Quando una classe non eredita da un'altra, si parla convenzionalmente di ***applicazione***, mentre quando eredita dalla classe '**`java.applet.Applet`**' (cioè da '`java/applet/Applet.class`') si usa la definizione ***applet***.

Contenuto della classe

La classe contiene essenzialmente dichiarazioni di variabili e metodi. L'esecuzione di un metodo dipende da una chiamata, detta anche ***messaggio***. Perché una classe si traduca in un programma autonomo, occorre che al suo interno ci sia un metodo che viene eseguito in modo automatico all'avvio.

Nel caso delle classi che non ereditano nulla da altre, come nell'esempio, ci deve essere il metodo '**`main`**' che viene eseguito all'avvio del binario Java contenente la classe stessa. Quando una classe eredita da un'altra, queste regole sono stabilite dalla classe ereditata.

Il metodo '**`main`**' è formato necessariamente come nell'esempio: '**`public static void main(String[] args) {...}`**'.

Variabili e tipi di dati

In Java si distinguono fondamentalmente due tipi di rappresentazione dei dati: primitivi e riferimenti a oggetti. I tipi di dati primitivi sono per esempio i soliti tipi numerici (intero, a virgola mobile, ecc.); gli altri sono ***oggetti***. Un oggetto è quindi una variabile contenente un riferimento a una struttura, più o meno complessa. In Java, gli array e le stringhe sono oggetti; pertanto non esistono tipi di dati primitivi equivalenti.

I nomi delle variabili possono essere composti utilizzando caratteri Unicode. Naturalmente, non è possibile utilizzare nomi coincidenti con parole chiave già utilizzate dal linguaggio stesso. La convenzione stilistica di Java richiede che il nome delle variabili inizi con la lettera minuscola; inoltre, se si tratta di un nome composto, la convenzione richiede di segnalare l'inizio di ogni nuova parola con una lettera maiuscola. Per esempio: '`miaVariabile`', '`dataOdierna`', '`elencoNomiFemminili`'.

Chiamata per valore

In Java, le chiamate dei metodi avvengono trasferendo il valore degli argomenti indicati nella chiamata stessa. Ciò significa che le modifiche che si dovessero apportare all'interno dei metodi non si riflettono all'indietro. Tuttavia, questo ragionamento vale solo per i tipi di dati primitivi, dal momento che quando si utilizzano degli oggetti, essendo questi dei riferimenti, le variazioni fatte al loro interno rimangono anche dopo la chiamata.

Variabili e tipi di dati

«

Si è già accennato al fatto che Java distingue tra due tipi di dati, primitivi e riferimenti a oggetti (o più semplicemente solo oggetti). L'esempio seguente mostra la dichiarazione di un intero all'interno di un metodo e il suo incremento fino a raggiungere un valore predefinito:

```
/**
 *  DieciXApp.java
 *  Un esempio di utilizzo delle variabili.
 */

import java.lang.*; // predefinita

class DieciXApp
{
    public static void main (String[] args)
    {
        int contatore = 0;

        // Inizia un ciclo in cui si emettono 10 «x» attraverso lo
        // standard output.
        while (contatore < 10)
          {
            contatore++;
            System.out.println ("x"); // emette una «x»
          }
    }
}
```

Tipi

«

I tipi di dati primitivi rappresentano un valore singolo. Il loro elenco si trova nella tabella u123.4.

Tabella u123.4. Elenco dei tipi di dati primitivi in Java.

Tipo	Dimensione	Descrizione
`byte`	8 bit, complemento a due.	Intero a 8 bit.
`short`	16 bit, complemento a due.	Intero ridotto.
`int`	32 bit, complemento a due.	Intero normale.
`long`	64 bit, complemento a due.	Intero molto grande.
`float`	32 bit	Virgola mobile, singola precisione.
`double`	64 bit	Virgola mobile, doppia precisione.
`char`	16 bit, carattere Unicode.	Carattere.
`boolean`	*Vero* o *Falso*.	Valore booleano.

Nell'esempio mostrato precedentemente, viene dichiarato un intero normale, '**`contatore`**', inizializzato al valore zero, che poi viene incrementato all'interno di un ciclo:

```
        int contatore = 0;

        // Inizia un ciclo in cui si emettono 10 «x» attraverso lo
        // standard output.
        while (contatore < 10)
        {
            contatore++;
            System.out.println ("x"); // emette una «x»
        }
```

«

Ogni tipo primitivo ha la possibilità di essere rappresentato in forma di costante letterale. La tabella u123.6 mostra l'elenco dei tipi di dati abbinati alla rappresentazione in forma di costante letterale.

Tabella u123.6. Elenco dei tipi di dati primitivi abbinati a una possibile rappresentazione in forma di costante letterale.

Tipo	Esempio di costante	Descrizione o intervallo
`byte`	123	−128..+127
`short`	12345	−32768..+32767
`int`	1234567890	$-(2^{31})..+((2^{31})-1)$
`long`	123456789012345678**90**	$-(2^{63})..+((2^{63})-1)$
`float`	(float)123.456	La costante con virgola è sempre a doppia precisione.
`double`	123.456	
`char`	'A'	Si usano gli apici semplici.
`boolean`	true	Si usano le parole chiave '**`true`**' e '**`false`**'.

È importante osservare che una costante numerica a virgola mobile è sempre a doppia precisione, per cui, se si vuole assegnare a una variabile a singola precisione ('**`float`**') una costante letterale, occorre una conversione di tipo, per mezzo di un cast. In seguito vengono descritte le stringhe, che si delimitano utilizzando gli apici doppi. Per ora è solo il caso ti tenere in considerazione che in Java le stringhe

non sono tipi di dati primitivi, ma oggetti veri e propri.

Campo di azione

Il campo di azione delle variabili in Java viene determinato dalla posizione in cui queste vengono dichiarate. Ciò determina il momento della loro creazione e distruzione. A fianco del concetto del campo di azione, si pone quello della ***protezione***, che può limitare l'accessibilità di una variabile. La protezione viene analizzata in seguito.

A seconda del loro campo di azione, si distinguono in particolare tre categorie più importanti di variabili: variabili appartenenti alla classe (*member variable*), variabili locali e parametri dei metodi.

Variabili appartenenti alla classe

Queste variabili appartengono alle classi e come tali sono dichiarate all'interno delle classi stesse, ma all'esterno dei metodi. L'esempio seguente mostra la dichiarazione della variabile '**`serveAQualcosa`**' come parte della classe '**`FaQualcosa`**'.

```
class FaQualcosa
{
    int serveAQualcosa = 0;

    // Dichiarazione dei metodi
    ...
}
```

Variabili locali

Sono variabili dichiarate all'interno dei metodi. Vengono create alla chiamata del metodo e distrutte alla sua conclusione. Per questo sono visibili solo all'interno del metodo che le dichiara.

Nell'esempio visto in precedenza, quello che visualizza 10 «x», la variabile '**`contatore`**' veniva dichiarata all'interno del metodo '**`main`**'.

Parametri dei metodi

Le variabili indicate in concomitanza con la dichiarazione di un metodo (quelle che appaiono tra parentesi tonde), vengono create nel momento della chiamata del metodo stesso e distrutte alla sua conclusione. Queste variabili contengono la copia degli argomenti utilizzati per la chiamata; in questo senso si dice che le chiamate ai metodi avvengono per valore.

Operatori ed espressioni

«

Gli operatori sono qualcosa che esegue un qualche tipo di funzione, su uno o due operandi, restituendo un valore. Il valore restituito è di tipo diverso a seconda degli operandi utilizzati. Per esempio, la somma di due interi genera un risultato intero.

Gli operandi descritti nelle sezioni seguenti sono solo quelli più comuni e importanti. In particolare, sono stati omessi quelli necessari al trattamento delle variabili in modo binario.

Operatori aritmetici

«

Gli operatori che intervengono su valori numerici sono elencati nella tabella u123.8.

Tabella u123.8. Elenco degli operatori aritmetici e di quelli di assegnamento relativi a valori numerici.

Operatore e operandi	Descrizione
++*op*	Incrementa di un'unità l'operando prima che venga restituito il suo valore.
op++	Incrementa di un'unità l'operando dopo averne restituito il suo valore.
−−*op*	Decrementa di un'unità l'operando prima che venga restituito il suo valore.
op−−	Decrementa di un'unità l'operando dopo averne restituito il suo valore.
+*op*	Non ha alcun effetto.
−*op*	Inverte il segno dell'operando.
op1 + *op2*	Somma i due operandi.
op1 − *op2*	Sottrae dal primo il secondo operando.
op1 ⋆ *op2*	Moltiplica i due operandi.
op1 / *op2*	Divide il primo operando per il secondo.
op1 % *op2*	Modulo -- il resto della divisione tra il primo e il secondo operando.
var = *valore*	Assegna alla variabile il valore alla destra.
op1 += *op2*	*op1* = *op1* + *op2*
op1 −= *op2*	*op1* = *op1* − *op2*

Operatore e operandi	Descrizione
op1 `*=` ***op2***	***op1*** `=` ***op1*** `*` ***op2***
op1 `/=` ***op2***	***op1*** `=` ***op1*** `/` ***op2***
op1 `%=` ***op2***	***op1*** `=` ***op1*** `%` ***op2***

Operatori di confronto e operatori logici

«

Gli operatori di confronto determinano la relazione tra due operandi. Il risultato dell'espressione composta da due operandi posti a confronto è di tipo booleano, rappresentabile in Java dalle costanti letterali '**`true`**' e '**`false`**'. Gli operatori di confronto sono elencati nella tabella u123.9.

Tabella u123.9. Elenco degli operatori di confronto. Le metavariabili indicate rappresentano gli operandi e la loro posizione.

Operatore e operandi	Descrizione
op1 `==` ***op2***	*Vero* se gli operandi si equivalgono.
op1 `!=` ***op2***	*Vero* se gli operandi sono differenti.
op1 `<` ***op2***	*Vero* se il primo operando è minore del secondo.
op1 `>` ***op2***	*Vero* se il primo operando è maggiore del secondo.
op1 `<=` ***op2***	*Vero* se il primo operando è minore o uguale al secondo.

Operatore e operandi	Descrizione
op1 `>=` ***op2***	*Vero* se il primo operando è maggiore o uguale al secondo.

Quando si vogliono combinare assieme diverse espressioni logiche, comprendendo in queste anche delle variabili che contengono un valore booleano, si utilizzano gli operatori logici (noti normalmente come: AND, OR, NOT, ecc.). Il risultato di un'espressione logica complessa è quello dell'ultima espressione elementare che sia stata valutata effettivamente. Gli operatori logici sono elencati nella tabella u123.10.

Tabella u123.10. Elenco degli operatori logici. Le metavariabili indicate rappresentano gli operandi e la loro posizione.

Operatore e operandi	Descrizione
`!` ***op***	Inverte il risultato logico dell'operando.
op1 `&&` ***op2***	Se il risultato del primo operando è *Falso* non valuta il secondo.
op1 `||` ***op2***	Se il risultato del primo operando è *Vero* non valuta il secondo.

Concatenamento di stringhe

Si è accennato al fatto che in Java, le stringhe siano oggetti e non tipi di dati primitivi. Esiste tuttavia la possibilità di indicare stringhe letterali nel modo consueto, attraverso la delimitazione con gli apici doppi.

Diverse stringhe possono essere concatenate, in modo da formare una stringa unica, attraverso l'operatore '**+**'.

```
    public static void main (String[] args)
    {
        int contatore = 0;

        while (contatore < 10)
          {
            contatore++;
            System.out.println ("Ciclo n. " + contatore);
          }
    }
```

Nel pezzo di codice appena mostrato, appare in particolare l'istruzione seguente:

```
            System.out.println ("Ciclo n. " + contatore);
```

L'espressione '**"Ciclo n. " + contatore**' si traduce nel risultato seguente:

```
Ciclo n. 1
Ciclo n. 2
…
Ciclo n. 10
```

In pratica, il contenuto della variabile '**contatore**' viene convertito automaticamente in stringa e unito alla costante letterale precedente.

Strutture di controllo del flusso

«

Le strutture di controllo del flusso delle istruzioni sono molto simili a quelle del linguaggio C. In particolare, dove può essere messa un'istruzione si può mettere anche un gruppo di istruzioni delimitate dalle parentesi graffe.

Normalmente, le strutture di controllo del flusso basano questo controllo sulla verifica di una condizione espressa all'interno di

parentesi tonde.

Struttura condizionale: «if»

«

```
if (condizione) istruzione
```

```
if (condizione) istruzione else istruzione
```

Se la condizione si verifica, viene eseguita l'istruzione (o il gruppo di istruzioni) seguente; quindi il controllo passa alle istruzioni successive alla struttura. Se viene utilizzato '**else**', nel caso non si verifichi la condizione, viene eseguita l'istruzione che ne segue. Vengono mostrati alcuni esempi.

```
int importo;
…
if (importo > 10000000) System.out.println ("L'offerta è vantaggiosa");
```

```
int importo;
int memorizza;
…
if (importo > 10000000)
  {
    memorizza = importo;
    System.out.println ("L'offerta è vantaggiosa");
  }
else
  {
    System.out.println ("Lascia perdere");
  }
```

```
int importo;
int memorizza;
…
if (importo > 10000000)
  {
    memorizza = importo;
    System.out.println ("L'offerta è vantaggiosa");
  }
else if (importo > 5000000)
  {
    memorizza = importo;
    System.out.println ("L'offerta è accettabile");
  }
else
  {
    System.out.println ("Lascia perdere");
  }
```

Struttura di selezione: «switch»

«

L'istruzione '**`switch`**' è un po' troppo complessa per essere rappresentata in modo chiaro attraverso uno schema sintattico. In generale, l'istruzione '**`switch`**' permette di **saltare** a una certa posizione della struttura, in base al risultato di un'espressione. L'esempio seguente mostra la visualizzazione del nome del mese, in base al valore di un intero:

```
int mese;
…
switch (mese)
  {
    case 1: System.out.println ("gennaio"); break;
    case 2: System.out.println ("febbraio"); break;
    case 3: System.out.println ("marzo"); break;
    case 4: System.out.println ("aprile"); break;
    case 5: System.out.println ("maggio"); break;
    case 6: System.out.println ("giugno"); break;
    case 7: System.out.println ("luglio"); break;
    case 8: System.out.println ("agosto"); break;
    case 9: System.out.println ("settembre"); break;
    case 10: System.out.println ("ottobre"); break;
    case 11: System.out.println ("novembre"); break;
    case 12: System.out.println ("dicembre"); break;
  }
```

Come si vede, dopo l'istruzione con cui si emette il nome del mese attraverso lo standard output, viene aggiunta un'istruzione di salto '**`break`**', che serve a uscire dalla struttura, perché altrimenti le istruzioni del caso successivo, se c'è, verrebbero eseguite. Infatti, un gruppo di casi può essere raggruppato assieme, quando si vuole che questi eseguano lo stesso gruppo di istruzioni:

```
int mese;
int giorni;
…
switch (mese)
  {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        giorni = 31;
        break;
    case 4:
    case 6:
```

```
            case 9:
            case 11:
                giorni = 30;
                break;
            case 2:
                if (((anno % 4 == 0) && !(anno % 100 = 0))
                    || (anno % 400 == 0))
                    giorni = 29;
                else
                    giorni = 28;
                break;
          }
```

È anche possibile definire un caso predefinito che si verifichi quando nessuno degli altri si avvera:

```
        int mese;
        …
        switch (mese)
          {
            case 1: System.out.println ("gennaio"); break;
            case 2: System.out.println ("febbraio"); break;
            …
            case 11: System.out.println ("novembre"); break;
            case 12: System.out.println ("dicembre"); break;
            default: System.out.println ("mese non corretto"); break;
         }
```

Iterazione con condizione di uscita iniziale: «while»

«

```
while (condizione) istruzione
```

‘**`while`**’ esegue un’istruzione, o un gruppo di queste, finché la condizione restituisce il valore *Vero*. La condizione viene valutata prima di eseguire il gruppo di istruzioni e poi ogni volta che termina un ciclo, prima dell’esecuzione del successivo. Segue il pezzo dell’e-

sempio già visto, di quella classe che visualizza 10 volte la lettera «x»:

```
        int contatore = 0;

        while (contatore < 10)
          {
            contatore++;
            System.out.println ("x");
          }
```

Nel blocco di istruzioni di un ciclo '**`while`**', ne possono apparire alcune particolari:

- '**`break`**'

  esce definitivamente dal ciclo '**`while`**';

- '**`continue`**'

  interrompe l'esecuzione del gruppo di istruzioni e riprende dalla valutazione della condizione.

L'esempio seguente è una variante del ciclo di visualizzazione mostrato sopra, modificato in modo da vedere il funzionamento di '**`break`**'. Si osservi che '**`while (true)`**' equivale a un ciclo senza fine, perché la condizione è sempre vera:

```
        int contatore = 0;

        while (true)
          {
            if (contatore >= 10)
              {
                break;
              }
            contatore++;
            System.out.println ("x");
          }
```

Iterazione con condizione di uscita finale: «do-while»

```
do blocco_di_istruzioni while (condizione);
```

'**do**' esegue un gruppo di istruzioni una volta e poi ne ripete l'esecuzione finché la condizione restituisce il valore *Vero*.

Iterazione enumerativa: «for»

```
for (espressione1; espressione2; espressione3) istruzione
```

Questa è la forma tipica di un'istruzione '**for**', in cui la prima espressione corrisponde all'assegnamento iniziale di una variabile, la seconda a una condizione che deve verificarsi fino a che si vuole che sia eseguita l'istruzione (o il gruppo di istruzioni), mentre la terza serve per l'incremento o decremento della variabile inizializzata con la prima espressione. In pratica, potrebbe esprimersi nella sintassi seguente:

```
for (var = n; condizione; var++) istruzione
```

Il ciclo '**for**' potrebbe essere definito anche in maniera differente, più generale: la prima espressione viene eseguita una volta sola all'inizio del ciclo; la seconda viene valutata all'inizio di ogni ciclo e il gruppo di istruzioni viene eseguito solo se il risultato è *Vero*; l'ultima viene eseguita alla fine dell'esecuzione del gruppo di istruzioni, prima che si ricominci con l'analisi della condizione.

Il vecchio esempio banale, in cui veniva visualizzata per 10 volte una «x», potrebbe tradursi nel modo seguente, attraverso l'uso di un

ciclo ‘**for**’:

```
int contatore;

for (contatore = 0; contatore < 10; contatore++)
  {
    System.out.println ("x");
  }
```

Array e stringhe

In Java, array e stringhe sono oggetti. In pratica, la variabile che contiene un array o una stringa è in realtà un riferimento alla struttura di dati rispettiva.

Array

La dichiarazione di un array avviene in Java in modo molto semplice, senza l’indicazione esplicita del numero di elementi. La dichiarazione avviene come se si trattasse di un tipo di dati normale, con la differenza che si aggiungono una coppia di parentesi quadre a sottolineare che si tratta di un array di elementi di quel tipo. L’esempio seguente dichiara che ‘**arrayDiInteri**’ è un array in cui gli elementi sono di tipo intero (‘**int**’), senza specificare quanti siano:

```
int[] arrayDiInteri;
```

Per fare in modo che l’array esista effettivamente, occorre che questo sia inizializzato, fornendogli gli elementi. Si usa per questo l’operatore ‘**new**’ seguito dal tipo di dati con il numero di elementi racchiuso tra parentesi quadre. L’esempio seguente assegna alla variabile ‘**arrayDiInteri**’ il riferimento a un array composto da sette interi:

```
arrayDiInteri = new int[7];
```

Nella pratica, è normale inizializzare l'array quando lo si dichiara; per cui, quanto già visto si può ridurre all'esempio seguente:

```
int[] arrayDiInteri = new int[7];
```

Il riferimento a un elemento di un array avviene aggiungendo al nome della variabile che rappresenta l'array stesso, il numero dell'elemento, racchiuso tra parentesi quadre. Come nel linguaggio C, il primo elemento si raggiunge con l'indice zero, mentre l'ultimo corrisponde alla dimensione meno uno.

Si è detto che gli array sono oggetti. In particolare, è possibile determinare la dimensione di un array, espressa in numero di elementi, leggendo il contenuto della variabile '**`length`**' dell'oggetto array. Nel caso dell'esempio già visto, si tratta di leggere il contenuto di '**`arrayDiInteri.length`**'.

L'esempio seguente mostra una scansione di un array, indicando una condizione di interruzione del ciclo indipendente dalla conoscenza anticipata della dimensione dell'array stesso. In particolare, la variabile '**`i`**' viene dichiarata contestualmente con la sua inizializzazione, nella prima espressione di controllo del ciclo '**`for`**':

```
for (int i = 0; i < arrayDiInteri.length; i++) {
    arrayDiInteri[i] = i;
}
```

Un array può contenere sia elementi primitivi che riferimenti a oggetti. In questo modo si possono avere gli array multidimensionali. L'esempio seguente rappresenta il modo in cui può essere definito un array 3x2 di interi e anche come scandirne i vari elementi:

```
/**
 *      Matrice3x2App.java
 *      Esempio di uso di array multidimensionali.
 */
```

```
import java.lang.*; // predefinita

class Matrice3x2App
{
    public static void main (String[] args)
    {
        int[][] matrice = new int[3][2];

        for (int i = 0; i < matrice.length; i++)
          {
            for (int j = 0; j < matrice[i].length; j++)
              {
                matrice[i][j] = 1000 + j + i * 10;
                System.out.println ("matrice[" + i + "][" + j + "] = "
                                    + matrice[i][j]);
              }
          }
    }
}
```

L'esecuzione di questo piccolo programma, genera il risultato seguente:

```
matrice[0][0] = 1000
matrice[0][1] = 1001
matrice[1][0] = 1010
matrice[1][1] = 1011
matrice[2][0] = 1020
matrice[2][1] = 1021
```

Stringhe

Le stringhe in Java sono oggetti e se ne distinguono due tipi: stringhe costanti e stringhe variabili. La distinzione è utile perché questi due tipi di oggetti hanno bisogno di una forma di rappresentazione diversa. Così, ciò porta a un'ottimizzazione del programma, che per una stringa costante richiede meno risorse rispetto a una stringa che deve essere variabile, oltre a migliorare altri aspetti legati alla sicurezza.

La dichiarazione di una variabile che possa contente un riferimento a un oggetto stringa-costante, si ottiene con la dichiarazione seguente:

```
String variabile;
```

In pratica, si dichiara che la variabile può contenere un riferimento a un oggetto di tipo '**String**'. La creazione di questo oggetto '**String**' si ottiene come nel caso degli array, utilizzando l'operatore '**new**'.

```
new String (stringa);
```

L'esempio seguente crea la variabile '**stringaCostante**' di tipo '**String**' e la inizializza assegnandoci il riferimento a una stringa:

```
String stringaCostante = new String ("Ciao ciao.");
```

Fortunatamente, si possono utilizzare anche delle costanti letterali pure e semplici. Per cui la stringa '**"Ciao ciao."**' è già di per sé un oggetto stringa-costante.

Si è già accennato al fatto che le stringhe-costanti possono essere concatenate facilmente utilizzando l'operatore '**+**':

```
"Ciao " + "come " + "stai?"
```

L'esempio restituisce un'unica stringa-costante, come quella seguente:

```
"Ciao come stai?"
```

Inoltre, in questi concatenamenti, entro certi limiti, possono essere inseriti elementi diversi da stringhe, come nell'esempio seguente, dove il contenuto numerico intero della variabile '**contatore**'

viene convertito automaticamente in stringa prima di essere emesso attraverso lo standard output.

```
int contatore = 0;

while (contatore < 10)
{
    contatore++;
    System.out.println ("Ciclo n. " + contatore);
}
```

Le stringhe variabili sono oggetti di tipo '**StringBuffer**' e vengono descritte più avanti.

Metodo «main()»

Si è accennato al fatto che una classe che non eredita esplicitamente da un'altra, richiede l'esistenza del metodo '**main()**' e viene detta applicazione. Questo metodo deve avere una forma precisa e si tratta di quello che viene chiamato automaticamente quando si avvia il binario Java corrispondente alla classe stessa. Senza questa convenzione, non ci sarebbe un modo per avviare un programma Java.

```
public static void main (String[] args) { istruzioni }
```

Nella sintassi indicata, le parentesi graffe fanno parte della dichiarazione del metodo e delimitano un gruppo di istruzioni.

args

È Importante osservare l'unico parametro del metodo '**main()**': l'array '**args**' composto da elementi di tipo '**String**'. Questo array contiene gli argomenti passati al programma Java attraverso la riga di comando.

L'esempio seguente, mostra come si può leggere il contenuto di questo array, tenendo presente che non si conosce inizialmente la sua dimensione. L'esempio emette separatamente, attraverso lo standard output, l'elenco degli argomenti ricevuti.

```
/**
 *  LeggiArgomentiApp.java
 *  Legge gli argomenti e gli emette attraverso lo standard output.
 */

import java.lang.*; // predefinita

class LeggiArgomentiApp
{
    public static void main (String[] args)
    {
        int i;

        for (i = 0; i < args.length; i++)
          {
            System.out.println (args[i]);
          }
    }
}
```