Fondamenti della sintassi di Java

Per utilizzare un linguaggio di programmazione è indispensabile conoscerne la sintassi, ovvero gli elementi di base e le regole per il loro utilizzo.

Istruzioni

Le *istruzioni* rappresentano in Java delle singole unità di esecuzione, un po' come le frasi del linguaggio naturale, che in qualche modo rappresentano singole unità; possono contenere chiamate di metodi, assegnazioni di variabili e così via. In Java ogni istruzione deve terminare con un punto e virgola, come nei seguenti esempi:

```
String testo = "Questa è una frase di testo";
int n = 10;
g.drawString("Ciao alla mia città!, 5, 25);
```

Blocchi

I *blocchi* sono gruppi di varie istruzioni correlate racchiuse tra parentesi graffe; in pratica rappresentano delle istruzioni composte e possono essere utilizzati ovunque possa stare una singola istruzione. Di seguito è riportato un esempio:

```
class Prova {
    public static void main (String args[]) {
        {
        int x = 10;
        int y = 50;
        System.out.println(x + y);
        }
    }
}
```

Commenti

In Java esistono tre modi per inserire dei commenti nel codice. Il primo modo consiste nell'utilizzare la doppia barra obliqua `//` su una riga; il testo che segue viene considerato un commento, come nel seguente esempio:

```
int numPagine = 192; // Un libro di 192 pagine
```

Il secondo modo consiste nel racchiudere una serie di righe di commento tra i caratteri `/*` e `*/`, come nel seguente esempio:

```
/* Questo è un commento che occupa più righe di
testo. Si tratta di un modo di specificare i commenti
che deriva direttamente da C e C++.
*/
```

Questo metodo è utile per inserire commenti su più righe.

Infine, è possibile racchiudere un testo tra i caratteri `/**` e `*/`. Questi codici di commento sono utilizzati dal sistema di generazione automatica della documentazione javadoc (per ulteriori informazioni al riguardo, si rimanda alla documentazione di Java che può essere consultata o prelevata dal sito dedicato agli sviluppatori).

Identificatori

Gli *identificatori* di Java (in pratica i nomi che identificano classi, metodi o variabili), sono sequenze con lunghezza virtualmente illimitata di caratteri Unicode, in cui il primo carattere deve essere una lettera o un trattino di sottolineatura (_) o un segno di dollaro ($), e non una cifra numerica.

Il set di caratteri Unicode consente di utilizzare i caratteri dei più comuni alfabeti internazionali, comprese le accentate e i segni particolari.

Per quanto riguarda gli identificatori, in Java vale la distinzione tra lettere maiuscole e minuscole, perciò `Casa` è diverso da `CASA` e anche da `casa`.

Parole chiave

Le seguenti parole chiave sono riservate per l'utilizzo interno del linguaggio, e non possono essere utilizzate come identificatori:

abstract	do	import	public	transient
boolen	double	instanceof	return	try
break	else	int	short	void
byte	extends	interface	static	volatile
case	final	long	strictfp	while
catch	finally	native	super	
char	float	new	switch	
class	for	null	synchronized	
const	goto	package	this	
continue	if	private	throw	
default	implements	protected	throws	

Le parole chiave `const` e `goto` sono riservate ma attualmente non sono utilizzate nel linguaggio.

`true`, `false` e `null` non sono parole chiave, anche se lo sembrano, ma letterali (i letterali sono trattati più avanti).

Variabili

Le *variabili* rappresentano locazioni di memoria a cui si può assegnare un nome, un tipo e un valore.

Prima di poter assegnare un valore a una variabile, è necessario dichiararla per definirne il tipo e il nome, come nei seguenti esempi:

```
String unTesto;
int numeroPagine;
boolean stato;
```

I tipi di dati che si possono utilizzare per le variabili in Java vengono trattati più avanti.

Per migliorare la leggibilità del codice, si consiglia di riportare le dichiarazioni delle variabili all'inizio, quando è possibile. In questo modo è più facile ritrovare subito il punto in cui sono definiti i tipi di dati utilizzati nel codice.
Al momento della dichiarazione di una variabile è anche possibile assegnare direttamente un valore, come nei seguenti esempi:

```
int numero = 45;
String testo = "Frase di esempio";
boolean stato = false;
```

Dichiarazioni e assegnamenti di variabili dello stesso tipo possono essere specificati su un'unica riga, separandoli con delle virgole, come nel seguente esempio:

```
int altezza = 187, peso = 85;
```

In Java esistono *variabili istanza*, *variabili di classe* e *variabili locali*. Le variabili istanza, già citate in precedenza, sono utilizzate per definire gli attributi degli oggetti; le variabili di classe definiscono invece gli attributi della classe, che non variano da un oggetto all'altro, e infine le variabili locali vengono utilizzate all'interno dei metodi, per memorizzare valori particolari che assumono rilievo in un ambito circoscritto.

Nel caso delle variabili locali, è obbligatorio assegnare esplicitamente un valore; alle variabili istanza e di classe viene invece assegnato un valore di default, che dipende dal tipo.

Tipi di dati

Una variabile può essere dichiarata come classe, interfaccia, array, o con uno degli otto tipi di dati primitivi di Java, elencati nella Tabella 7.1.

Tabella 7.1 I tipi di dati primitivi di Java.

Tipo di dati	Descrizione
byte	Numero intero a 8 bit (da -128 a 127).
short	Numero intero a 16 bit (da -32768 a 32767).
int	Numero intero a 32 bit (da -2147483648 a 2147483647).
long	Numero intero a 64 bit (da –9223372036854775808 a 9223372036854775807).
float	Numero in virgola mobile a 32 bit.

Tabella 7.1 I tipi di dati primitivi di Java.

Tipo di dati	Descrizione
doub le	Numero in virgola mobile a 64 bit.
char	Stringa di caratteri Unicode a 16 bit.
boolean	Valore logico `true` o `false`.
void	Utilizzato come tipo di ritorno.
Object	Oggetto.

Letterali

I *letterali* sono valori considerati in sé, a cui non viene assegnato un nome. In Java ne esistono di cinque tipi: intero, a virgola mobile, booleano, carattere e stringa.

Letterali carattere

I letterali carattere sono costituiti da un singolo carattere racchiuso tra apici singoli, come ad esempio `'1'`, `'a'`, `'#'` e così via. Possono anche essere rappresentati da una sequenza di codici speciali dette *sequenze di escape* (le sequenze più comuni sono elencate nella Tabella 7.2, dove *c* rappresenta una cifra).

Tabella 7.2 Sequenze di escape per caratteri speciali.

Sequenza	Significato
\n	Nuova riga.
\t	Tabulazione.
\b	Backspace (cancella il carattere a sinistra).
\r	Ritorno a capo.
\f	Nuova pagina.
\\	Barra inversa.
\'	Apice singolo.
\"	Apice doppio.
ccc	Ottale
\x*cc*	Esadecimale
\u*ccc*	Carattere Unicode

Letterali stringa

I letterali stringa sono serie di caratteri, e in Java sono considerati oggetti `String`. In questo senso si distinguono dagli altri tipi di letterali, che non sono considerati oggetti a nessun titolo.

Anche la stringa vuota, di lunghezza zero, è considerata un letterale valido. Di seguito sono riportati alcuni esempi di letterali stringa.

```
"Frase di testo"
"" // Questa è la stringa vuota
```

Letterali interi e a virgola mobile

I letterali interi possono essere rappresentati in base 8 (ottali), 10 (decimali) o 16 (esadecimali).

I letterali ottali iniziano con uno zero e contengono cifre da 0 a 7, quelli decimali non hanno lo zero iniziale e comprendono cifre da 0 a 9, quelli esadecimali iniziano con uno zero seguito da X o x (0X oppure 0x) e contengono cifre da 0 a 9 o lettere dalla "a" alla "f" (o dalla "A" alla "F"). Di seguito sono riportati alcuni esempi:

```
042       // Ottale
123       // Decimale
0x12a5    // Esadecimale
```

I letterali a virgola mobile sono costituiti da una parte intera in base 10, un punto decimale (come in tutti i linguaggi di programmazione internazionali, si utilizza il punto decimale al posto della virgola decimale), un esponente (`e` o `E` seguito da un intero con segno) e un suffisso che ne specifica il tipo. Il suffisso può essere `f` o `F` nel caso dei letterali di tipo `float`, `d` o `D` nel caso dei letterali di tipo `double`.

Di seguito sono riportati alcuni esempi:

```
2.3
2.3f
2.3F
2.3e10
2.3E10
2.3d
2.3D
2.34De11
```

Letterali booleani

I letterali booleani sono soltanto due: `true` (vero, 1) e `false` (falso, 1).

Operatori

Gli *operatori* sono simboli speciali utilizzati per eseguire delle operazioni. Sono elencati nella Tabella 7.3.

Tabella 7.3 Gli operatori di Java.

Sequenza	Significato
+	Somma.
-	Sottrazione.
*	Moltiplicazione.
/	Divisione.
%	Modulo.
<	Minore di.
>	Maggiore di.
<=	Minore o uguale a.
>=	Maggiore o uguale a.
==	Uguale a.
!=	Diverso da.
&&	AND logico.
||	OR logico.
!	NOT logico.
&	AND.
^	XOR.
|	OR.
=	Assegnamento.
++	Incremento.
--	Decremento.
+=	Somma e assegnamento.
-=	Sottrazione e assegnamento.
*=	Moltiplicazione e assegnamento.
/=	Divisione e assegnamento.
%=	Modulo e assegnamento

Separatori

I caratteri elencati di seguito sono utilizzati come *separatori*:

```
(   )   {   }   [   ]   ;   ,   .
```

Espressioni

Le *espressioni* sono sequenze di variabili e operatori utilizzati per eseguire determinate operazioni restituendo un risultato. Le operazioni possono essere di tipo aritmetico, oppure confronti logici, o assegnamenti. Per quanto riguarda l'utilizzo degli operatori all'interno delle espressioni, è necessario seguire precise regole di priorità, che determinano l'ordine in cui le gli operatori stessi vengono valutati. Ad esempio, l'espressione:

```
n = 5 + 3 * 2;
```

produce il risultato 11, perché l'operatore di moltiplicazione (*) viene valutato prima di quello di addizione (+).

Per ignorare le regole di priorità degli operatori è possibile utilizzare le parentesi, come nel seguente esempio:

```
n = (5 + 3) * 2;
```

In questo caso il risultato è 16, perché la porzione di espressione racchiusa tra parentesi viene valutata sempre per prima. Nella Tabella 7.4 sono elencati gli operatori di Java ordinati per priorità (quelli sulla prima riga hanno la priorità massima).

Tabella 7.4 Gli operatori di Java ordinati per valore di priorità.

```
. [] ()
++ -- ! ~ instanceof
* / %
+ -
<< >> >>>>
< > <= >=
== !=
&
^
¦
&&
¦¦
? :
= += -= *= /= &= ¦= ^= %= >>= <<= >>>>=
,
```

Operazioni aritmetiche

Java dispone degli operatori necessari per eseguire le cinque operazioni aritmetiche fondamentali: addizione (+), sottrazione (-), moltiplicazione (*), divisione (/) e modulo (%).

Tutti questi operatori sono *binari*, nel senso che richiedono due *operandi*: uno a sinistra e uno a destra dell'operatore, come nel seguente esempio:

```
5 * 3
```

Nelle operazioni sui numeri interi, il tipo di dati del risultato può essere `int` o `long`, non `byte`, `short` o `char`. Se entrambi gli operandi sono di tipo `int` il risultato è di tipo `int`, se uno è di tipo `long`, anche il risultato è di tipo `long`.

Incremento e decremento

Gli operatori ++ e -- incrementano e decrementano rispettivamente un valore di un'unità. Di seguito sono riportati alcuni esempi:

```
n = 5;
n++;        // n ora vale 6
x=10;
x--;        // x ora vale 9
```

Questi operatori sono *unari*, nel senso che agiscono su un unico operando. Tale operando può essere specificato prima o dopo dell'operatore, e il risultato può cambiare a seconda dei casi. In effetti, nel caso in cui si utilizzano gli operatori prefissi negli assegnamenti, possono verificarsi strane situazioni. Le espressioni:

```
y = x++;
```

e:

```
y = ++x;
```

possono fornire risultati diversi, poiché nel primo caso il valore di x viene assegnato a y e poi incrementato, mentre nel secondo viene prima incrementato e poi assegnato a y. In pratica, se x inizialmente vale 1, dopo l'esecuzione della prima espressione il valore di x è 2 e il valore di y è 1 (y riceve il valore di x prima che quest'ultimo sia incrementato). Nel secondo caso, dopo l'esecuzione dell'espressione x vale 2 e y vale anch'essa 2 (il valore di x viene assegnato a y dopo essere stato incrementato).

Assegnamento

L'operatore di assegnamento fondamentale è il segno di uguale (=), che assegna all'operando di sinistra il valore specificato a destra. Esistono poi degli operatori composti, elencati di seguito, che insieme all'assegnamento eseguono un'operazione aritmetica o logica.

```
a += b;     // Equivale ad a = a + b
a -= b;     // Equivale ad a = a - b
a *= b;     // Equivale ad a = a * b
a /= b;     // Equivale ad a = a / b
a &= b;     // Equivale ad a = a & b
a ¦= b;     // Equivale ad a = a ¦ b
a ^= b;     // Equivale ad a = a ^ b
```

È possibile concatenare più assegnamenti sulla stessa riga, come nel seguente esempio:

```
a = b = c = 5;
```

Confronto

Gli operatori di confronto consentono di verificare se due stringhe coincidono o se due numeri sono uguali. Il risultato del confronto è sempre un valore booleano `true` o `false`. Di seguito sono riportati alcuni esempi:

```
a == b      // verifica se a è uguale a b
a != b      // verifica se a è diverso da b
a < b       // verifica se a è minore di b
a > b       // verifica se a è maggiore di b
a <= b      // verifica se a è minore o uguale a b
a >= b      // verifica se a è maggiore o uguale a b
```

Operazioni logiche

Esiste la possibilità di eseguire operazioni sui valori booleani `true` e `false`, mediante gli operatori logici di Java. Di seguito sono riportati alcuni esempi:

```
!a          // Negazione di a
a & b       // Restituisce true se e soltanto se
            // a e b valgono entrambi true
a ¦ b       // Restituisce true se e soltanto se
            // a vale true oppure b vale true
a ^ b       // Restituisce true se e soltanto se
            // a e b hanno valori diversi
a && b      // Equivale ad a & b, con valutazione
            // abbreviata
a ¦¦ b      // Equivale ad a ¦ b, con valutazione
            // abbreviata
a ? b : c;  // Equivale a if...then...else
```

Operazioni sui bit

Gli operatori sui bit derivano da C e C++, e consentono di manipolare le rappresentazioni in bit dei numeri interi. Si tratta di un argomento avanzato, per il quale si rimanda alla documentazione di Java.

Operazioni sulle stringhe

L'operatore + consente di concatenare due stringhe di caratteri: il risultato è una singola stringa costituita da tutti i caratteri della prima seguiti da tutti i caratteri della seconda. Anche questo operatore può essere combinato con l'operatore di assegnamento (=).

Di seguito sono riportati alcuni esempi:

```
a = "Ciao" + " "+ "mamma";    // Il risultato è
                              // "Ciao mamma"
a += "e papà";                // Nuovo risultato:
                              // "Ciao mamma e papà"
```

Operazioni su oggetti

L'operatore `instanceof` consente di verificare se un oggetto è un'istanza di una classe particolare, restituendo `true` o `false` a seconda dei casi. Di seguito è riportato un esempio:

```
a = pianoforte instanceof Chitarra;

/* a riceve il valore false, poiché
un pianoforte non può essere istanza
della classe Chitarra */
```

Altri elementi fondamentali

In questo capitolo vengono esaminate le istruzioni condizionali, i cicli e gli array, tutti elementi indispensabili per la programmazione.

Istruzioni condizionali

Nei programmi si presenta spesso la necessità di eseguire una determinata azione in base al verificarsi di una particolare condizione. Le *istruzioni condizionali* consentono di regolare il flusso di esecuzione del programma secondo le necessità, ramificandolo in base al risultato di un confronto sul valore di una variabile o di una espressione. Le istruzioni condizionali di Java sono `if` e `switch`, entrambe descritte in dettaglio nel seguito.

L'istruzione if

Questa istruzione è presente in forma più o meno analoga in quasi tutti i linguaggi di programmazione, e si utilizza quasi sempre allo stesso modo; di seguito è riportato un esempio:

```
if (a == b) System.out.println("a è uguale a b");
else System.out.println("a non è uguale a b");
```

Notate che la struttura è del tipo `if (`*`condizione`*`)` *`istruzione`*, dove *`condizione`* è un'espressione che restituisce un risultato booleano e *`istruzione`* può essere qualsiasi istruzione Java, che viene eseguita se *`condizione`* restituisce `true`.

La parola chiave `else` è facoltativa e consente di specificare l'istruzione da eseguire se `condizione` vale `false`.

L'operatore condizionale

Java dispone inoltre di un particolare operatore, detto *operatore condizionale*, che equivale in pratica all'istruzione `if` ma restituisce un risultato: `true` o `false`. La sua sintassi è la seguente:

```
condizione ? seVero : seFalso;
```

`condizione` anche in questo caso deve restituire un valore booleano; se questo è `true`, l'operatore condizionale restituisce il valore *`seVero`*, altrimenti restituisce *`seFalso`*. Il vantaggio dell'operatore condizionale è che restituisce un valore, il quale può quindi essere assegnato a una variabile, come nel seguente esempio:

```
a = 5;
b = 10;
int n = a < b ? a : b;
```

In questo caso, alla variabile `n` viene assegnato il valore di `a`, poiché è vero che `a` è minore di `b`.

L'istruzione switch

Questa istruzione offre un'alternativa all'utilizzo dell'istruzione `if`, utile nel caso in cui si desideri ramificare il flusso di esecuzione del programma in base a diversi possibili valori assunti da una condizione. Considerate il seguente esempio:

```
switch (cond) {
    case 1:
        istr1;
        break;
        case 2:
        istr2;
        break;
    case 3:
        istr3;
        break;
    ...
    default:
        istrDefault;
    }
```

In questo caso il valore di `cond` viene confrontato con i valori specificati nelle clausole `case`; se `cond` vale 1, viene eseguita `istr1`, dopodiché si esce dal blocco (grazie a `break`). Se `cond` vale 2, viene eseguita `istr2` e così via. Se `cond` non coincide con nessuno dei valori specificati nelle clausole `case`, viene eseguita l'istruzione specificata dopo la clausola `default` (che è facoltativa: se manca, non viene eseguita alcuna istruzione).

Il valore specificato dopo `case` non deve necessariamente essere un numero: può essere una variabile o un'espressione, purché il risultato sia intero; l'istruzione `switch` infatti è limitata ai numeri interi, e l'unico tipo di confronto che è in grado di eseguire è l'uguaglianza.

Lo stesso risultato fornito dall'istruzione `switch` in realtà può essere ottenuto anche con `if`, ma in un modo leggermente più complicato, come nel seguente esempio:

```
if (cond == 1)
    istr1;
else if (cond == 2)
    istr2;
else if (cond == 3)
    istr3;
else
    istrDefault;
```

Cicli

Un'altra struttura fondamentale dei linguaggi di programmazione è rappresentata dai *cicli*, che consentono di ripetere l'esecuzione di una o più istruzioni per un numero fissato di volte, o fino a quando è verificata una particolare condizione. In Java esistono tre costrutti per i cicli: `for`, `while` e `do`.

I cicli for

I cicli `for` consentono di ripetere l'esecuzione di una o più istruzioni per un numero di volte specificato; la sintassi è:

```
for (inizio; condizione; incremento) {
    istruzioni;
}
```

All'avvio del ciclo viene eseguita l'espressione *`inizio`*, che solitamente attua l'assegnazione di un valore iniziale a una variabile utilizzata come contatore (ad esempio, `int i = 0`). Poi viene valutata la *`condizione`*, che solitamente esegue un controllo sul valore del contatore (ad esempio, `i < 5`), e se questa restituisce `true` vengono eseguite le *`istruzioni`* (che possono essere più di una); dopodiché viene valutata l'espressione *`incremento`*, che solitamente esegue l'incremento del contatore, e il ciclo riprende valutando nuovamente la *`condizione`* ed eseguendo nuovamente le *`istruzioni`*. Il ciclo termina quando *`condizione`* è `false`.

Per chiarire meglio le cose, ecco un esempio concreto:

```
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

In questo caso il contatore è la variabile *`i`*, a cui inizialmente viene assegnato il valore 0, e che viene incrementata di 1 a ogni ripetizione del ciclo. Il ciclo termina quando il valore di `i` raggiunge 5. Il risultato è illustrato nella Figura 8.1.

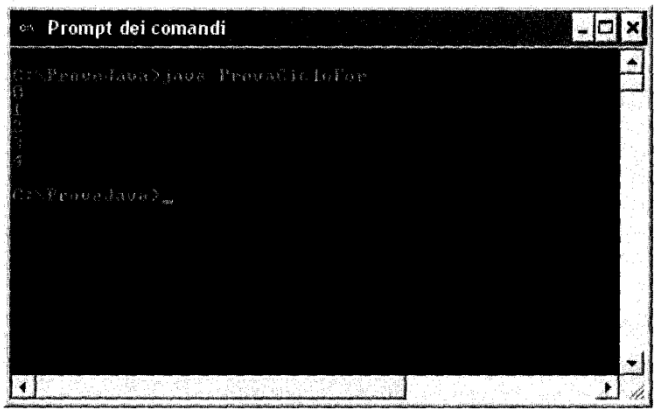

Figura 8.1 Il risultato dell'esecuzione di un ciclo for.

I cicli while

Questi cicli consentono di ripetere l'esecuzione di una o più istruzioni in base al valore di una condizione specificata. La sintassi è:

```
while (condizione) {
    istruzioni;
}
```

Il ciclo inizia valutando la *condizione*: se questa risulta `true`, vengono eseguite le *istruzioni* e il ciclo si ripete. Il ciclo termina quando *condizione* risulta `false`.

Di seguito è riportato un esempio concreto:

```
int numMese = 1;

int totAffitto = 0;

while (numMese <= 12) {
    totAffitto += Affitto(numMese);
    numMese++;
}
```

In questo caso viene calcolato l'affitto pagato in un anno, sommando tutti gli importi dei pagamenti mensili, calcolati dal metodo `Affitto()`.

Il ciclo termina quando il valore di `numMese` supera 12, poiché in un anno ci sono soltanto 12 mesi.

I cicli do

Questi cicli sono simili ai cicli `while`, con la differenza che in questo caso la condizione viene valutata soltanto dopo l'esecuzione delle istruzioni.

La sintassi è:

```
do {
    istruzioni;
} while (condizione);
```

In questo modo, le *istruzioni* vengono eseguite almeno una volta, anche se *condizione* risulta subito `false`.

Ecco un esempio:

```
int n = 10;
do {
    System.out.println(n);
    n--;
} while (n < 10 & n > 0);
```

Il risultato dell'esecuzione del ciclo è illustrato nella Figura 8.2. Notate che inizialmente la condizione non è vera, ma le istruzioni vengono eseguite comunque. Se si fosse utilizzato un ciclo `while`, le istruzioni non sarebbero mai state eseguite e sullo schermo non sarebbe stato visualizzato alcun valore.

Array

Gli array in Java sono oggetti della classe `Object`, di cui possono richiamare tutti i metodi. Sono costituiti da una serie di variabili, le *componenti*, che devono essere tutte dello stesso tipo.

Le componenti dell'array sono disposte una dopo l'altra in sequenza, ed è possibile modificarle e inserirne di nuove, oppure eliminarle.

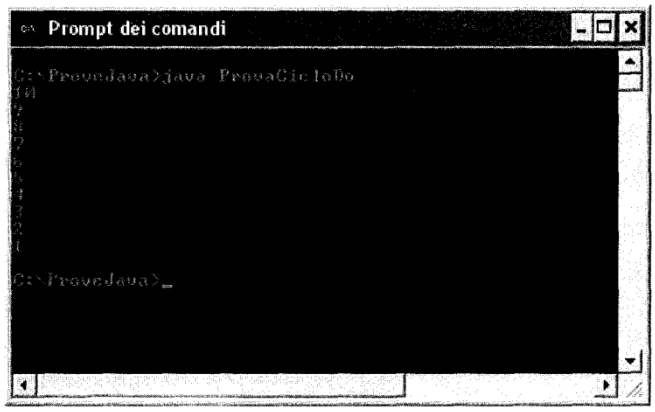

Figura 8.2 Il risultato dell'esecuzione di un ciclo do.

Per creare un array occorre dichiarare una variabile che lo contenga, creare un oggetto array e assegnarlo alla variabile, e infine inserire gli elementi che costituiscono le componenti dell'array. Di seguito è riportato un esempio:

```
String[] indirizzi = new String[10];
```

Questa istruzione crea un array di stringhe denominato `indirizzi`, contenente 10 elementi.

La coppia di parentesi quadre che contrassegna la dichiarazione di un array può essere specificata dopo la dichiarazione di tipo o dopo il nome della variabile; entrambe le forme sono accettate.

In pratica, l'istruzione riportata in precedenza è equivalente alla seguente:

```
String indirizzi[] = new String[10];
```

Gli elementi di un array sono identificati da un *indice*, ovvero un numero che indica la loro posizione all'interno dell'array.

Per accedere a un elemento basta specificarne l'indice, come nel seguente esempio:

```
System.out.println(indirizzi[0]);
```

Questa istruzione visualizza sullo schermo il contenuto del primo elemento dell'array `indirizzi`.

La numerazione delle posizioni all'interno di un array inizia da 0, come in C e C++.

Se si specifica una posizione che oltrepassa i limiti di definizione dell'array, fissati al momento della dichiarazione, si ottiene un errore durante la compilazione o l'esecuzione del programma.

È possibile modificare gli elementi di un array assegnando loro un nuovo valore, come nel seguente esempio:

```
indirizzi[0] = "Via Volta 26";
indirizzi[1] = "Corso Milano 50";
```

Tutti gli array dispongono di una variabile istanza denominata `length` che ne indica la lunghezza (ovvero il numero di elementi).

Così, la seguente istruzione visualizza il numero di elementi contenuti nell'array `indirizzi`:

```
System.out.println(indirizzi.length);
```

Ulteriori informazioni sull'utilizzo delle variabili istanza di un oggetto sono riportate nel Capitolo 9.

Java non supporta gli array multidimensionali veri e propri, ma consente di creare array i cui elementi sono a loro volta array, con un risultato analogo, come nel seguente esempio:

```
int coppie[] [] = new int[8][8];
coppie[0][0] = 1;
coppie[0][1] = 2;
coppie[1][0] = 1;
coppie[1][1] = 2;
...
```