Lezione 7
Stringhe e array: un primo approccio agli oggetti

Java distingue due grandi categorie di dati: i tipi predefiniti, oramai completamente descritti, e gli oggetti, che da questo punto in avanti sono il principale argomento di studio del corso. I primi, inglobati direttamente nel cuore del linguaggio, coprono le esigenze di calcolo più elementari e costituiscono le fondamenta di ogni programma. Senza di essi, non sarebbe possibile progettare ed utilizzare delle strutture più complesse. Gli oggetti, al contrario, possono essere intesi come dei tipi personalizzati, che estendono le possibilità già integrate nel linguaggio. Ogni sviluppatore può e deve definire gli schemi di funzionamento di nuovi oggetti. Questi schemi sono le classi. Ad ogni modo, non è necessario reinventare ogni volta la ruota. Si supponga che per lo sviluppo di un certo programma sia stato necessario definire un nuovo tipo di dati. Lo stesso, all'occorrenza, può essere impiegato in altri progetti, semplicemente riutilizzando la classe Java o il bytecode (il file *.class* già compilato) che lo definiscono. La piattaforma Java 2 arriva allo sviluppatore con una nutrita schiera di oggetti pronti all'uso, ordinatamente riposti in spazi gerarchici chiamati *pacchetti*. In questo modo, il programmatore non deve sviluppare in proprio una classe dedicata, ad esempio, alla rappresentazione di date ed orari, giacché all'interno della piattaforma ne trova una già pronta e sperimentata.

Le *stringhe* e gli *array* sono due particolari modelli di dati. Non sono tipi predefiniti, sono degli oggetti, eppure possono avvantaggiarsi di caratteristiche di cui tutti gli altri oggetti non possono disporre. Per certi versi, le stringhe e gli array sono oggetti profondamente integrati nel linguaggio stesso.

7.1 - Stringhe
Un basilare approccio alle stringhe è già stato svolto nel corso delle lezioni precedenti. Creare un oggetto di tipo *String* (questo modo di dire, d'ora in avanti, sarà sempre più ricorrente) è semplice:

```
String miaStringa = "Ciao";
```

Anche gli operatori di concatenazione sono stati completamente descritti (cfr. **Lezione 5**). Ciò che delle stringhe ancora non è noto, riguarda le caratteristiche che più le avvicinano alla programmazione orientata agli oggetti.

Ogni oggetto, in Java, dispone di *proprietà* e *metodi*. Le *proprietà* forniscono informazioni sullo stato corrente dell'oggetto, mentre i *metodi* sono funzioni che permettono all'oggetto di svolgere un'azione. A livello teorico, questa breve ed inefficiente definizione sarà dettagliatamente esaminata ed approfondita nel corso della prossima lezione. Per il momento, è più interessante capire come, nella pratica, i metodi e le proprietà di un oggetto possano essere sfruttate a vantaggio dei propri software.

Ogni oggetto di tipo *String* dispone di una nutrita schiera di metodi. Qualche linea di codice, in casi come questo, vale più di mille altre precisazioni:

```
// Creo un oggetto di tipo String
String miaStringa = "Ciao";
// Richiamo il metodo charAt() per sapere
// quale carattere apra la stringa. In Java,
// il primo carattere di una stringa ha indice 0,
```

```
// il secondo ha indice 1, il terzo ha indice 2
// e così via.
char primo = miaStringa.charAt(0);
System.out.println(primo); // Stampa "C"
// Voglio recuperare una sottostringa formata
// dai primi tre caratteri della stringa contenuta
// nell'oggetto miaStringa
String sottoStringa = miaStringa.substring(0, 3);
System.out.println(sottoStringa); // Stampa "Cia"
// Voglio sapere l'indice associato al carattere 'i'
// presente in miaStringa
int indice = miaStringa.indexOf('i');
System.out.println(indice); // Stampa "1"
```

Questo esempio ha dimostrato l'uso di tre metodi di cui ogni oggetto di tipo *String* è dotato: *charAt()*, *indexOf()* e *substring()*. Quando si effettua una chiamata del tipo

```
miaStringa.indexOf('i')
```

si osserva il generico modello, basato sulla notazione puntata, mostrato di seguito:

```
riferimento.nomeMetodo(argomenti)
```

Il riferimento, come si vedrà bene in seguito, è la variabile che lavora da ancora verso l'oggetto. La terminologia ufficiale chiama *oggetto di invocazione* l'oggetto sul quale è stato attivato un metodo.
Non è detto che un metodo debba necessariamente richiedere degli argomenti. In casi come questo, è necessario includere ugualmente la coppia di parentesi tonde che seguono il nome del metodo, alla seguente maniera:

```
riferimento.nomeMetodoSenzaArgomenti()
```

Alcuni metodi svolgono esclusivamente delle operazioni all'interno dell'oggetto di invocazione, altri restituiscono un risultato al codice chiamante. Il risultato restituito dai metodi dotati di tale possibilità è detto *valore di ritorno*. Come è lecito attendersi, ciascun valore di ritorno è di uno specifico tipo che è interessante conoscere. In tal modo, i valori di ritorno possono essere letti, memorizzati ed utilizzati come più conviene. Ad esempio, come si è visto nel codice presentato poco sopra, il metodo *indexOf()* restituisce un valore di tipo *int*, *charAt()* restituisce un *char* e *substring()* un nuovo oggetto *String*.

L'elenco dei metodi di cui sono dotate le stringhe è davvero folto e ben nutrito. La seguente tabella fornisce una panoramica dei metodi di uso più comune:

Tipo restituito	***Metodo e parametri***	***Descrizione***
`int`	`charAt(int i)`	Restituisce il carattere alla posizione *i*.
`boolean`	`endsWith(String s)`	Restituisce *true* se l'oggetto di invocazione termina con la sottostringa *s*.
`boolean`	`equals(String s)`	Restituisce *true* quando l'oggetto di invocazione e *s* rappresentano la medesima sequenza di caratteri.

Tipo restituito	*Metodo e parametri*	*Descrizione*
`int`	`indexOf(char c)`	Restituisce la prima posizione del carattere *c*, oppure -1 nel caso tale carattere non faccia parte della stringa.
`int`	`indexOf(char c, int i)`	Come il precedente, con la differenza che la ricerca del carattere *c* prende piede dalla posizione i.
`int`	`indexOf(String s)`	Restituisce la prima posizione della sottostringa *s*, oppure -1 nel caso tale sottostringa non compaia nell'oggetto di invocazione.
`int`	`indexOf(String s, int i)`	Come il precedente, con la differenza che la ricerca della sottostringa *s* prende piede dalla posizione i.
`int`	`length()`	Restituisce la lunghezza della stringa.
`String`	`replace(char c1, char c2)`	Restituisce una nuova stringa, ottenuta dall'oggetto di invocazione sostituendo il carattere *c2* ad ogni occorrenza del carattere *c1*.
`boolean`	`startsWith(String s)`	Restituisce *true* se l'oggetto di invocazione inizia con la sottostringa s.
`String`	`toLowerCase()`	Restituisce una nuova stringa, ottenuta traslando verso il minuscolo ogni carattere dell'oggetto di invocazione.
`String`	`toUpperCase()`	Restituisce una nuova stringa, ottenuta traslando verso il maiuscolo ogni carattere dell'oggetto di invocazione.
`String`	`trim()`	Restituisce una nuova stringa, ottenuta dall'oggetto di invocazione eliminando gli spazi che precedono il primo carattere significativo e quelli che seguono l'ultimo. In pratica, *" ciao ".trim()* restituisce *"ciao"*.

Tabella 7.1
I più importanti metodi di cui sono dotati gli oggetti di tipo *String*.

Come è possibile osservare dalla tabella, gli oggetti di Java ammettono più metodi con il medesimo nome, purché differiscano per quel che riguarda gli argomenti processati.

7.2 - Array

Un *array* è un gruppo di variabili del medesimo tipo, cui ci si riferisce con un nome comune ed un indice numerico. A differenza di C e C++, dove gli array sono spazi di memoria allocati in aree adiacenti, gli array di Java sono in tutto e per tutto degli oggetti. Un array monodimensionale può essere creato osservando il seguente modello:

```
nomeTipo[] nomeArray = new nomeTipo[dimensione];
```

"tipo[] nomeArray" o "tipo nomeArray[]"?
Java supporta due differenti sintassi per la dichiarazione di un array. La prima, esaminata nel **Paragrafo 7.2**, è quella logicamente più corretta, così strutturata:

```
tipo[] nomeArray
```

Ad esempio:

```
int[] mioArray
```

La seconda forma, del tutto equivalente alla prima nel risultato prodotto, pone la coppia di parentesi quadre a fianco del nome dell'array, anziché del suo tipo:

```
tipo nomeArray[]
```

Ad esempio:

```
int mioArray[]
```

La forma logicamente più corretta, come si è anticipato, è la prima, giacché in Java gli array sono oggetti, e dunque la loro natura è più strettamente legata al tipo che non al nome. Diversamente avviene in C, dove la seconda forma è quella corretta, giacché gli array di C sono aree di memoria affiancate, e non oggetti. Java supporta ambo le notazioni per facilitare la stesura del codice a chi già è abituato alle forme di C/C++.

Ad esempio:

```
String[] giorniDellaSettimana = new String[7];
```

Questo codice genera un array monodimensionale, costituito da sette elementi di tipo *String*.

La dimensione di un array può essere determinata staticamente, mediante una costante di qualsiasi tipo, oppure dinamicamente, servendosi di un valore intero noto solo al momento dell'esecuzione:

```
// Dimensionamento statico
int[] arr1 = new int[2];

// Dimensionamento dinamico
// Si supponga che n sia una variabile di tipo numerico intero
int[] arr2 = new int[n];
```

Una volta dichiarato ed inizializzato un array, è possibile accedere ai diversi elementi che ne fanno parte servendosi del particolare operatore *[ ]*:

```
String[] giorniDellaSettimana = new String[7];

giorniDellaSettimana[0] = "Lunedì";
giorniDellaSettimana[1] = "Martedì";
giorniDellaSettimana[2] = "Mercoledì";
giorniDellaSettimana[3] = "Giovedì";
giorniDellaSettimana[4] = "Venerdì";
giorniDellaSettimana[5] = "Sabato";
giorniDellaSettimana[6] = "Domenica";

for (int i = 0; i < 7; i++)
  System.out.println(giorniDellaSettimana[i]);
```

Il primo elemento di un array di dimensione *n* ha indice 0, mentre l'ultimo ha indice *n* - 1.

Gli array possono avvantaggiarsi di una forma breve, per l'inizializzazione in linea degli elementi che lo costituiscono:

```
String[] giorniDellaSettimana = {
  "Lunedì", "Martedì", "Mercoledì", "Giovedì",
  "Venerdì", "Sabato", "Domenica"
};
```

Questa forma è spesso usata quando si è alle prese con un array di ridotte dimensioni.

Gli array di Java dispongono di una particolare proprietà, ossia l'intero *length*, che riporta la dimensione dell'array:

```
String[] giorniDellaSettimana = new String[7];
System.out.println(giorniDellaSettimana.length); // Stampa "7"
```

Un ciclo *for* ideato allo scopo di scorrere gli elementi di un array, pertanto, può essere facilmente scritto al seguente modo:

```
for (int i = 0; i < nomeArray.length; i++) {
  // Operazioni su nomeArray[i]
}
```

In lingua italiana, gli array monodimensionali vengono chiamati *vettori*. Java, ad ogni modo, supporta anche il concetto di array multidimensionale o *matrice*:

```
int[][] matrice = new int[2][3];
matrice[0][0] = 2;
matrice[0][1] = 4;
matrice[0][2] = 7;
matrice[1][0] = 0;
matrice[1][1] = 5;
matrice[1][2] = 3;
```

Questo codice genera una matrice di dimensione 2 × 3, costituita da sei elementi. La **Figura 7.1** fornisce una rappresentazione grafica della struttura.

	colonna 0	colonna 1	colonna 2
riga 0	2	4	7
riga 1	0	5	3

```
matrice[0][0] = 2;
matrice[0][1] = 4;
matrice[0][2] = 7;
matrice[1][0] = 0;
matrice[1][1] = 5;
matrice[1][2] = 3;
```

Figura 7.1
Rappresentazione grafica di una matrice di 2 righe e 3 colonne.

Una matrice è rempre resa mediante un array di array. Nel codice sopra mostrato, ad esempio, la matrice è in realtà un vettore di dimensione 2, i cui elementi sono vettori di dimensione 3. Questo comporta che la proprietà *matrice.length* indica di quante righe è composta la matrice, mentre *matrice[0].length* riporta il numero di colonne presenti lungo la prima riga. Il seguente codice riproduce sulla linea dei comandi l'ultima figura mostrata:

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

Creare matrici a tre o più dimensioni è altrettanto semplice:

```
char[][][] treDimensioni = new char[4][2][4];

String[][][][] quattroDimensioni = new String[2][3][2][4];
```

Ogni ragionamento svolto in merito alle matrici bidimensionali può essere facilmente esteso e adattato ai nuovi casi introdotti.

Inizializzazione automatica degli elementi di un array

Quando si inizializza un array, si inizializzano automaticamente anche tutti i suoi elementi. Ad esempio, scrivendo

```
int[] arr = new int[5];
```

si ottengono cinque interi impostati sul valore 0. Semplificando, è come se fosse stato scritto:

```
int[] arr = {0, 0, 0, 0, 0};
```

Lo stesso avviene con i tipi numerici *byte*, *char*, *short*, *long*, *float* e *double*. I tipi *boolean* vengono automaticamente posizionati su *false*. I riferimenti agli oggetti, invece, vengono inizializzati con lo speciale valore *null*. E' possibile verificare l'osservazione eseguendo il codice:

```
String[] s = new String[5];
for (int i = 0; i < s.length; i++)
      System.out.println(s[i]);
```

Il significato, gli scopi e le conseguenze dei valori *null* saranno esaminati nella prossima lezione.

ArrayIndexOutOfBoundsException

Il seguente codice, benché contenga un errore, viene normalmente compilato:

```
int[] i = new int[5];
System.out.println(i[5]);
```

Gli indici degli array, lo si ricorda, possono spaziare da 0 a *n* - 1, dove con *n* si intende la dimensione del vettore. Dunque, *i[5]* non è un elemento valido, giacché l'ultimo elemento del vettore dichiarato è *i[4]*. Il codice, comunque sia, è sintatticamente corretto, e per questo può essere compilato senza proteste. Il problema verrà a galla a tempo di esecuzione, quando si riceverà il messaggio di errore:

```
java.lang.ArrayIndexOutOfBoundsException
```