Capitolo 3

Strutture di controllo decisionali

Andrea Gini

Dopo aver introdotto il concetto di variabile e di array, è giunto il momento di analizzare a fondo i restanti costrutti del linguaggio Java. Come si è già visto nell'introduzione, i costrutti fondamentali di un linguaggio di programmazione sono quelli decisionali e quelli iterativi. Il linguaggio Java prevede tre costrutti decisionali e tre iterativi: in questo capitolo verranno analizzate in profondità le strutture di controllo del primo tipo. I programmatori C troveranno familiari i costrutti di Java: questo infatti riprende la sintassi del C introducendo variazioni minime.

Una struttura di controllo decisionale permette al programmatore di vincolare l'esecuzione di un'istruzione (o di un blocco di istruzioni) a una condizione booleana. Prima di analizzare l'essenza di tali costrutti, è bene chiarire cosa si intenda esattamente con i termini "espressione booleana" e "blocco di istruzioni".

Condizioni booleane

Una condizione booleana è un'espressione della quale si può dire se sia vera o falsa (in inglese, true o false). Nell'introduzione sono stati già illustrati alcuni semplici esempi, che fanno uso degli operatori di uguaglianza, maggiore e minore. In questa sede vale la pena di approfondire la possibilità di combinare le condizioni booleane mediante gli operatori logici AND, OR, NOT e XOR.

In Java l'operatore AND viene rappresentato dal carattere &. L'AND logico opera su due parametri, e restituisce `true` solamente se entrambi sono `true`. Se si desidera che l'istruzione x = x + 1 venga eseguita solo se il valore della variabile x è maggiore di 10 e contemporaneamente minore di 100 (ossia compreso tra 10 e 100) si può scrivere:

```
if(x >= 10 && x <= 100)
  x = x + 1;
```

Si noti che nell'esempio l'operatore & viene ripetuto due volte: questa variante dell'AND, denominata short circuit, segnala al calcolatore che può interrompere la valutazione dell'espressione non appena sia stata verificato il suo valore di verità, migliorando l'efficienza di esecuzione (per esempio, se durante la valutazione dell'espressione il calcolatore scopre che il primo parametro di una AND è falso, tutta l'espressione risulterà falsa indipendentemente dal valore del secondo parametro).

L'operatore OR, rappresentato in Java con il carattere |, restituisce true se uno o entrambi i parametri sono veri, mentre restituisce false solamente quando entrambi i parametri sono falsi. Pertanto, se si desidera che l'istruzione x = x * 2 venga eseguita solo se x è uguale a 7 o a 8, si dovrà scrivere:

```
if ( x == 7 || x == 8 )
  x = x * 2;
```

Anche in questo caso, si è fatto ricorso all'operatore short circuit || al fine di rendere più efficiente la valutazione: se il primo parametro è vero, l'espressione è vera indipendentemente dal valore del secondo parametro.

L'operatore XOR (OR esclusivo), rappresentato in Java con il carattere ^, restituisce true solo se uno dei due parametri è vero e l'altro falso; se al contrario i parametri sono entrambi veri o entrambi falsi, l'espressione restituisce false. Si noti che non esiste un operatore short circuit per l'OR esclusivo, dal momento che è necessario valutare entrambi i parametri per fornire un valore di verità.

Infine, l'operatore NOT (in Java il carattere !) permette di negare una qualsiasi espressione, restituendo in tal modo un valore true se l'espressione è false e viceversa.

È possibile scrivere espressioni complicate a piacere, combinando tra loro un numero qualsiasi di espressioni più semplici e ricorrendo alle parentesi per rendere esplicite le precedenze. La seguente istruzione azzera la variabile x se il suo valore è compreso tra 10 e 20 o tra 30 e 40, estremi inclusi:

```
if((x>=10 && x <= 20) || (x >= 30 && x <= 40))
  x = 0;
```

Blocco di istruzioni e variabili locali

Negli esempi visti fino a ora, l'istruzione if è stata usata per vincolare l'esecuzione di un'unica istruzione. Come ci si deve comportare se si desidera vincolare un numero maggiore di istruzioni? In casi come questi si deve definire un blocco, ossia un insieme di istruzioni racchiuso tra parentesi graffe, che il compilatore Java tratta come un'istruzione unica. Pertanto, se si desidera azzerare le variabili x, y e z qualora una di esse superi il valore 100, si può scrivere un frammento di codice del tipo:

```
if ( x >= 100 || y >= 100 || z >= 100 ) {
  // se la condizione è vera, tutte le
  // seguenti istruzioni verranno eseguite
  x = 0;
  y = 0;
  z = 0;
}
```

Per convenzione, quando si apre una parentesi graffa, le righe successive vengono fatte rientrare di un paio di spazi. Questa pratica, che prende comunemente il nome di indentazione, è del tutto arbitraria: niente impedisce di riscrivere il frammento di codice precedente in questo modo:

```
if ( x >= 100 || y >= 100 || z >= 100 )
{ x = 0; y = 0; z = 0; }
```

Come si è già visto in altre occasioni, le convenzioni di impaginazione aiutano a rendere il codice più leggibile, e di conseguenza più facile da correggere o da mantenere.

Una particolarità dei blocchi è che al loro interno è possibile definire variabili locali. Tali variabili hanno una ciclo di vita ridotto, che termina non appena il flusso di esecuzione esce dal blocco.

Pertanto, in un caso come il seguente:

```
if ( x !=  y ) {
  int t = x;
  x = y;
  y = t;
}
t = 0; // ERRORE!
```

l'ultima istruzione è errata perché fa riferimento alla variabile `t`, definita all'interno del precedente blocco e che al di fuori di esso ha cessato di esistere.

if – else

Il costrutto condizionale più usato in Java è l'`if`, che può essere usato nelle due varianti con o senza `else`. Il primo tipo, mostrato nel seguente esempio:

```
if ( condizioneBooleana )
  istruzione;
```

esegue l'istruzione se la condizione booleana è vera, mentre prosegue senza fare niente in caso contrario. La variante con l'`else` ha una forma del tipo:

```
if ( condizioneBooleana )
  istruzione1;
else
  istruzione2;
```

e permette di specificare, oltre all'istruzione da eseguire in caso di successo, anche quella da eseguire in caso di fallimento. Come è stato già spiegato nel paragrafo precedente, se si desidera che venga eseguita più di un'istruzione è necessario ricorrere ai blocchi:

```
if ( condizioneBooleana ) {
  istruzione1a;
  istruzione2a;
  istruzione3a;
}
else {
  istruzione1b;
  istruzione2b;
  istruzione3b;
}
```

if – else annidati

Il costrutto if può comparire anche all'interno di un altro costrutto if, creando strutture nidificate anche molto complesse. Si osservi un frammento di codice con due if concatenati:

```
if( x >= 0 )
  if( x <= 10 )
    System.out.println("x è compreso tra 0 e 10");
```

Il primo di questi dice "se la variabile x è maggiore o uguale a 0, esegui l'istruzione seguente"; l'istruzione successiva è a sua volta un if che dice "se la variabile x è minore o uguale a cento, esegui l'istruzione successiva": pertanto la terza istruzione verrà eseguita solamente se entrambe le condizioni precedenti risultano vere.

Se si inserisce un else dopo queste istruzioni, a quale dei due if farà riferimento? Nel linguaggio Java, un'istruzione else fa sempre riferimento all'ultimo if della catena (quello più interno). Per sottolineare il concetto, si usa allineare l' else al corrispondente if:

```
if( x >= 0 )
  if( x <= 10 )
    System.out.println("x è compreso tra 0 e 10");
  else
    System.out.println("x è maggiore di 10");
```

Se ora si aggiunge un ulteriore else, esso farà riferimento al primo if:

```
if( x >= 0 )
  if( x <= 10 )
    System.out.println("x è compreso tra 0 e 10");
  else                  // riprende l'istruzione if( x <= 10 )
    System.out.println("x è maggiore di 10");
else                                          // riprende l'istruzione if( x >= 0 )
  System.out.println("x è minore di 0");
```

Come si può fare se si desidera forzare un else a fare riferimento a un if esterno? È possibile rimuovere dall'esempio precedente il primo else, in modo che quello che rimane faccia ancora riferimento all'if più esterno? Per ottenere questo effetto è necessario racchiudere l'if più interno in un blocco; in questo modo, l'if interno verrà trattato come un'istruzione a se stante, priva di else:

```
if( x >= 0 ) {
  if( x <= 10 )
    System.out.println("x è compreso tra 0 e 10");
}
else                                  // riprende l'istruzione if( x >= 0 )
  System.out.println("x è minore di 0");
```

È buona norma evitare di ricorrere pesantemente alla nidificazione di istruzioni if, data la confusione che spesso ne segue.

Durante la formulazione di combinazioni condizionali troppo complesse può capitare di commettere errori molto difficili da riconoscere. Con un po' di ragionamento è possibile formulare un'espressione più leggibile ricorrendo agli operatori booleani. Per esempio, il frammento di codice:

```
if( x >= 0 )
  if( x <= 10 )
    System.out.println("x è compreso tra 0 e 10");
```

può essere tranquillamente sostituito dal seguente, in tutto equivalente:

```
if( x >= 0 && x <= 10 )
  System.out.println("x è compreso tra 0 e 10");
```

if – else concatenati

Un caso più semplice di combinazione condizionale si ha quando si fa seguire un if a un else. In questo caso, la verità dell'espressione booleana presente in ciascun if viene rafforzata dalla condizione di verità di tutte le condizioni if precedenti, premettendo di creare una catena di alternative:

```
if( x <= 0 )
  System.out.println("x è minore o uguale a 0");
```

```
else if( x <= 10)
  System.out.println("x è maggiore di 0 e minore o uguale a 10");
else if ( x <= 20)
  System.out.println("x è maggiore di 10 e minore o uguale a 20");
else
  System.out.println("x è maggiore di 20");
```

Si noti che quando si utilizza una serie di if - else concatenati, l'istruzione else che compare alla fine andrà a coprire tutti i casi non considerati dalle precedenti condizioni.

Il costrutto switch – case

Il costrutto switch permette di gestire tutte quelle situazioni in cui si deve seguire un percorso differente a seconda del valore di un'espressione.

```
switch (espressione) {
  case val1:
    istruzione_1a;
    istruzione_2a;
    ....
    istruzione_na;
    break;
  case val2:
    istruzione_1b;
    istruzione_2b;
    ....
    istruzione_nb;
    break;
  default:
    istruzione_1default;
    istruzione_2defefault;
    ....
    istruzione_ndefault;
    break;
}
```

L'espressione contenuta tra le parentesi dello switch deve essere di tipo intero (int, byte, short o char); ogni istruzione case lavora su un particolare valore, e fornisce una sequenza di istruzioni da eseguire in quella particolare circostanza. Tale sequenza termina usualmente con l'istruzione break, che forza il computer a uscire dallo switch senza verificare i valori successivi. Nonostante il break sia opzionale, il suo uso è fortemente consigliato.

Dopo aver specificato un numero qualsiasi di case, si può chiudere l'elenco specificando il blocco di default, ossia una sequenza di istruzioni da eseguire se non si è verificato nessuno dei casi precedenti. Il blocco di default è opzionale, e pertanto verrà inserito solamente nelle circostanze nelle quali risulti necessario.

Il seguente esempio stampa uno specifico messaggio se x vale 1, 2 o 3, mentre stamperà un messaggio di default in tutti gli altri casi.

```
switch (x) {
  case 1:
    System.out.println("x è uguale a 1");
    break;
  case 2:
    System.out.println("x è uguale a 2");
    break;
  case 3:
    System.out.println("x è uguale a 3");
    break;
  default:
    System.out.println("x è diverso da 1, 2 e 3");
    break;
}
```

Espressioni condizionali

All'interno delle espressioni aritmetiche è possibile utilizzare l'operatore ?, che permette di rendere condizionale l'assegnamento di valore. Il costrutto si compone di tre parti: un'espressione booleana, seguita da un carattere ?, e due espressioni generiche separate a loro volta da un carattere :, come mostrato qui di seguito:

```
espressioneBooleana ? espressione1 : espressione2;
```

Quando l'espressione viene valutata, il calcolatore verifica il valore di verità dell'espressione booleana: se risulta vera, l'espressione restituisce il valore della prima espressione; in caso contrario restituisce il valore della seconda. Ecco un esempio di istruzione che assegna alla variabile y il valore assoluto di x, ossia il valore x se x è positivo e -x se x è negativo:

```
y = x < 0 ? -x : x;
```

Ovviamente, è sempre possibile costruire una forma equivalente ricorrendo ad un costrutto if - else:

```
if ( x < 0 )
  y = -x;
else
  y = x;
```

La scelta di una forma o dell'altra deve sempre privilegiare la leggibilità del codice risultante.