Capitolo 1

Tipi di dati

Andrea Gini

Nell'introduzione è stato introdotto il concetto di assegnamento su variabile intera. Il linguaggio Java offre altri tipi di variabile su cui lavorare: quattro tipi per gli interi, due per i numeri floating point, uno per i caratteri e uno per le variabili booleane. Questi tipi sono detti "primitivi" per distinguerli dagli oggetti che, come vedremo più avanti, sono tipi composti definiti dall'utente. Ogni tipo primitivo è una struttura algebrica composta da un insieme numerico e da un set di operazioni definite su di esso. I tipi primitivi si prestano a un uso intuitivo, che trascende la loro implementazione; esistono tuttavia delle situazioni limite in cui questo approccio non funziona: per evitare comportamenti inattesi è necessario conoscere le proprietà e i limiti di ciascun tipo.

Tipi interi

L'insieme degli interi, la somma e la sottrazione sono concetti che fanno parte del nostro bagaglio culturale fin dall'età prescolare. I calcolatori hanno una particolare predisposizione per i numeri interi, che possono essere trattati con grande efficienza e precisione assoluta; l'unico problema che può insorgere nel trattare tali numeri è dato dall'estensione del tipo che si utilizza, che non è mai infinita.

Una variabile di tipo `int`, per esempio, può contenere qualsiasi numero intero compreso tra 2.147.483.647 e -2.147.483.648, ossia tutti i numeri rappresentabili con una cella di memoria a 32 bit. Se il valore di una variabile supera la soglia più alta, in questo caso 2.147.483.647, essa ricomincia dal valore opposto, ossia da -2.147.483.648. È importante prestare grande attenzione a questo tipo di limitazioni quando si lavora su un calcolatore.

Il formato int è senza dubbio il tipo primitivo più usato, per ragioni di praticità e di efficienza: i 32 bit forniscono un ottimo compromesso tra l'estensione dell'insieme numerico e le prestazioni su tutte le moderne piattaforme hardware. Esistono comunque altri tre tipi interi, che si distinguono da int per la dimensione massima dei numeri trattabili. Il tipo byte permette di operare su numeri compresi tra -128 e 127, ossia i numeri che è possibile rappresentare con cifre da 8 bit. Malgrado la ridicola estensione dell'insieme sottostante, byte è un tipo di dato estremamente importante, dal momento che è il formato di base nello scambio di dati con le periferiche di input/output, come i dischi o la rete. Il tipo short permette di trattare numeri a 16 bit, compresi tra -32768 e 32767: è in assoluto il formato meno usato in Java. Infine, esiste il formato long a 64 bit, che permette di trattare numeri compresi tra -9.223.327.036.854.775.808 e 9.223.327.036.854.775.807. Malgrado l'estensione da capogiro, esso viene utilizzato esclusivamente nelle circostanze in cui risulti veramente utile: nei calcolatori attuali, quasi tutti a 32 bit, il tipo long viene trattato con minor efficienza rispetto a int. Il tipo long richiede inoltre una certa attenzione in fase di assegnamento: per assegnare valori superiori a quelli consentiti da una cifra a 32 bit, è necessario posporre al numero la lettera L:

```
long number = 4.543.349.547L;
```

Tabella 1.1 – *Tipi di dati interi.*

Tipo	Min	Max
byte	-128	127
short	-32768	32767
int	-2.147.483.648	2.147.483.647
long	-9.223.327.036.854.775.808	9.223.327.036.854.775.807

Tipi numerici floating point

Per elaborare numeri con decimali, Java mette a disposizione i tipi floating point (a virgola mobile). Il calcolo a virgola mobile funziona secondo il principio della notazione scientifica, dove un numero viene rappresentato mediante una parte intera, detta mantissa, moltiplicata per un'opportuna potenza (positiva o negativa) di 10:

$0{,}00000456 = 0{,}456 * 10^{-6}$
$345\ 675\ 432 = 0{,}345675432 * 10^{7}$

Questa modalità di rappresentazione offre il vantaggio di permettere la manipolazione di numeri molto più grandi o molto più piccoli di quanto sarebbe consentito dal numero di cifre disponibili, al prezzo di un arrotondamento per eccesso o per difetto:

0,00000000000000000007695439352 ≈ 0,769544 * 10^{-20}
576 469 830 453 324 239 437 892 544 ≈ 0,57647 * 10^{27}

I numeri in virgola mobile, malgrado la loro importanza nel calcolo scientifico, non vengono trattati diffusamente nei manuali di base. Le particolari modalità di arrotondamento e di perdita di precisione, caratteristici di questo tipo numerico, richiedono conoscenze matematiche non banali.

Il float, a 32 bit, può contenere numeri positivi e negativi compresi tra 1.40129846432481707 * 10^{-45} e 3.4282346638528860 * 10^{38}, mentre il double a 64 bit può lavorare su numeri positivi e negativi tra 4.94065655841246544 * 10^{-324} e 1.79769313486231570 * 10^{138}.

Tabella 1.2 – *Tipi di dati floating point.*

Tipo	Min	Max
Float	1.40129846432481707 * 10^{-45}	3.4282346638528860 * 10^{38}
Double	4.94065655841246544 * 10^{-324}	1.79769313486231570 * 10^{138}

La maggior parte dei calcolatori moderni è ottimizzata per il calcolo floating point a doppia precisione. Le funzioni matematiche presenti nelle librerie Java usano quasi sempre il tipo double.

Assegnamento di variabili floating point

Per l'assegnamento di variabili floating point si ricorre ad una speciale formulazione della notazione esponenziale. Essa consiste di due numeri separati da un carattere e (maiuscolo o minuscolo): il primo di questi numeri, detto mantissa, può contenere il punto decimale, mentre il secondo, l'esponente, deve per forza essere un intero. Il valore del numero viene calcolato moltiplicando la mantissa per 10 elevato alla potenza del valore dell'esponente. Nell'assegnare una variabile float è necessario posporre la lettera F, come negli esempi:

```
Rappresentazione float:     1.56e3F;
Rappresentazione double:    5.23423e102;
```

Booleano

Una variabile booleana può assumere solamente due valori: true e false, che significano rispettivamente "vero" e "falso". Per operare su valori booleani, è necessario ricorrere a un'algebra particolare, detta algebra booleana, che denota un calcolo caratterizzato da proprietà molto diverse rispetto a quello degli interi. Nonostante l'apparente semplicità, l'algebra booleana ha una potenza enorme: per essere precisi, è l'unica algebra che un calcolatore è in grado di trattare

a livello hardware. La mappatura del calcolo algebrico su quello booleano è un tema estremamente affascinante, che può essere approfondito, a seconda della specifica area di interesse, su un manuale di logica matematica, di informatica teorica o di elettronica digitale; in questa sede verranno approfondite solamente le proprietà elementari dei principali operatori.

Assegnamento su variabili booleane

Una variabile booleana può assumere solamente i valori true e false; pertanto, il modo più semplice di effettuare un assegnamento consiste nel porre una variabile a uno di questi due valori:

```
boolean a = true;
boolean b = false;
```

Il ricorso agli operatori relazionali ==, !=, >, <, >= e <= permette di assegnare a una variabile booleana il valore di verità di un'espressione. Per esempio:

```
boolean b = (a == 10);
```

assegna a b il valore di verità dell'espressione a == 10, che sarà true se la variabile 'a' contiene il valore 10, false in caso contrario. Le espressioni booleane possono essere combinate tramite gli operatori logici ! (NOT) , & (AND), | (OR) e ^ (XOR). Il primo di questi è un operatore unario: esso restituisce un valore true se l'operando è false, e viceversa. Per esempio:

```
boolean a = false;
boolean b = !a;
```

la variabile b assume il valore opposto ad a, ossia true. L'operando & lavora su due operatori. Esso restituisce true solo se entrambi gli operatori sono true; in tutti gli altri casi restituisce false. L'operatore | lavora su due parametri: esso restituisce true se almeno uno dei due parametri è true (in altre parole, è false solo quando entrambi gli operatori sono false). Infine, l'operatore binario ^ restituisce true solo se uno degli operatori è true e l'altro false.

Tabella 1.3 – *Tavole di verità degli operatori booleani.*

Negazione logica	
A	!A
false	true
false	true

And logico		
A	B	A&B
false	false	false
false	true	false
true	false	false
true	true	true

Or logico		
A	B	A | B
false	false	false
false	true	true
true	false	true
true	true	true

Or esclusivo		
A	B	A^B
false	false	false
false	true	true
true	false	true
true	true	false

Caratteri

Una variabile di tipo char può contenere un carattere in formato Unicode. La codifica Unicode comprende decine di migliaia di caratteri, e include gli alfabeti più diffusi nel mondo. I valori da 0 a 127 corrispondono, per motivi di retro compatibilità, al set di caratteri ASCII. Una variabile char è, a tutti gli effetti, un intero a 16 bit privo di segno: essa può assumere qualsiasi valore tra 0 e 65535. Nell'effettuare assegnamenti, tuttavia, si preferisce ricorrere alle costanti carattere, che si dichiarano racchiudendo un singolo carattere tra apici, come si vede nell'esempio seguente:

```
char carattere1 = 'a';
char carattere2 = 'z';
```

Il simbolo \ ha il ruolo di carattere di escape: grazie a esso è possibile specificare come costante char alcuni caratteri che non sarebbe possibile specificare con la tastiera. La tavola 1.4 presenta l'elenco completo delle sequenze di escape valide.

Tabella 1.4 – *Sequenze di escape.*

Sequenze di escape	
\n	nuova linea
\r	a capo
\f	nuova pagina
\'	carattere apice
\"	carattere doppio apice
\\	carattere backslash
\b	backspace
\t	carattere di tabulazione

Promozioni e casting

Durante la stesura di un programma capita di dover spostare valori numerici tra variabili di tipo diverso, come tra int e long. Se la variabile di destinazione è più capace di quella di partenza l'operazione, che in questo caso prende il nome di promozione, avviene in modo del tutto trasparente, come negli esempi seguenti:

```
byte b = 100;
short s = b;        // promozione da byte a short
int i = s;          // promozione da short a int
long l = i;         // promozione da int a long
```

È possibile, ancorché sconsigliato, effettuare l'operazione inversa. Un valore definito in una variabile più capiente può essere forzato in una variabile di tipo inferiore, ma nell'operazione possono andare perse delle informazioni. Se si dispone di una variabile intera che contiene un valore di 257 e si prova a forzare un simile valore in una variabile di tipo byte, che per sua natura può contenere valori tra -128 e 127, essa assumerà valore 1. La perdita di informazioni, che dipende dalla particolare modalità di memorizzazione dei valori nei vari tipi, può avere effetti indesiderati, o addirittura drammatici.

Il 4 giugno 1996 a Kourou, nella Guyana Francese, il razzo Ariane 5 si sollevò dalla sua rampa di lancio per un volo di collaudo. Il viaggio inaugurale di Ariane stabilì senza dubbio un primato, dal momento che durò appena quaranta secondi, e terminò con una fragorosa esplosione, che fortunatamente non produsse danni a persone. L'imbarazzante episodio venne trasmesso in mondovisione, con gran dispetto per l'Agenzia Spaziale Europea, che sul progetto Ariane aveva messo in gioco la propria credibilità. Un team di esperti fu incaricato di indagare sul disastro; dopo un'attenta analisi, giunsero alla conclusione che la causa del fallimento era stato un errore software nel sistema di riferimento inerziale. Più precisamente, un numero floating point a 64 bit, relativo alla velocità orizzontale del razzo rispetto alla piattaforma, veniva convertito, mediante un'operazione di casting, in un intero a 16 bit. Non appena il valore superò la fatidica soglia di 32.768, l'operazione cominciò a produrre valori sballati, che mandarono in crisi il sistema di navigazione. Questa circostanza provocò l'attivazione del sistema di autodistruzione, che polverizzò il razzo prima che potesse perdere il controllo e precipitare chissà dove.

Lo sviluppo di Ariane aveva richiesto, nell'arco di un decennio, una spesa complessiva di circa 7 miliardi di dollari. Al momento del lancio, il razzo trasportava quattro satelliti per telecomunicazioni, del valore complessivo di circa 500 milioni di dollari. Una semplice operazione di casting, introdotta, a quanto pare, per discutibili motivi di ottimizzazione, produsse pertanto un danno economico spropositato, oltre a un incalcolabile danno di immagine. A completare il quadro, pare che il carico non fosse neppure stato assicurato, una circostanza che probabilmente fornì nuovi corollari al celebre elenco delle "Leggi di Murphy".

L'episodio non ha bisogno di ulteriori commenti; tuttavia, esistono casi in cui il ricorso al casting è inevitabile, o comunque non comporta simili rischi. Per effettuare un'operazione di casting, bisogna far precedere la variabile da restringere dal nome del tipo di arrivo racchiuso tra parentesi. Le seguenti righe mostrano un esempio inverso al precedente.

```
long l = 100;
int i = (int)l;          // cast da long a int
short s = (short)i;      // cast da int a short
byte b = (byte)s;        // cast da short a byte
```

Autoincremento e autodecremento

Il linguaggio Java ha ereditato dal C gli operatori di autoincremento, che in molti casi semplificano la sintassi delle espressioni di assegnamento. L'espressione:

```
x = x + 1;
```

può essere riscritta ricorrendo all'operatore ++, come nell'esempio seguente:

```
x++;
```

Allo stesso modo, l'espressione `x = x - 1` è equivalente a `x--`.

Se si desidera effettuare un incremento di valore superiore a 1, si può ricorrere all'operatore +=. L'espressione `x = x + 10` può essere riscritta come `x += 10`.

In modo simile, gli operatori +=, -=, *= e %= permettono di semplificare gli assegnamenti che fanno uso delle altre operazioni aritmetiche.

Gli operatori ++ e -- possono comparire sia prima sia dopo il simbolo di variabile. La differenza tra i due casi è abbastanza sottile: se l'operatore precede la variabile, essa viene dapprima incrementata e poi valutata; quando invece l'operatore segue la variabile, la valutazione avverrà prima dell'operazione di incremento. Nel seguente esempio:

```
x = 10;
y = ++x*2;
```

la variabile x viene incrementata prima che venga calcolato il valore di y, che in definitiva assume il valore 22. Al contrario, nell'esempio:

```
x = 10;
y = x++*2;
```

la variabile x viene prima valutata col valore 10, causando in tal modo l'assegnamento del valore 20 (ossia 10 * 2) alla variabile y; subito dopo, viene incrementata a 11.