Tipi di dato e variabili

Walter Didimo

Tipi di dato

I dati che possono essere manipolati in un programma si classificano in tipologie differenti, chiamate tipi di dato

Esistono due macro tipi di dato in Java:

- tipi riferimento
- tipi primitivi

Tipi riferimento

I tipi riferimento sono i tipi associati alle classi

- se esiste una classe di nome C, allora esiste un tipo riferimento di nome C
- una variabile riferimento di tipo C può contenere il riferimento ad un oggetto della classe C

Ad esempio, come già visto più volte, se *String* è il nome di una classe, si può definire una variabile riferimento di tipo *String* al modo:

String s

Il letterale null

Ogni variabile riferimento può anche contenere uno speciale valore, il valore null

- *null* rappresenta un *riferimento nullo*, ossia il riferimento a nessun oggetto
- *null* è un valore riferimento costante, quello che in termini tecnici si chiama un letterale riferimento

Uso di null

A cosa serve *null* ?

Capita spesso che, durante l'esecuzione di un programma, una variabile riferimento si trovi a non contenere alcun riferimento utile

- in questi casi è buona norma che alla variabile venga assegnato il letterale *null*
- la presenza di *null* in una variabile ci informa proprio che la variabile non contiene riferimenti utili

Invocazione di metodi e null

Non si possono invocare metodi sul riferimento *null*

Se si invoca un metodo su una variabile che contiene il valore *null*, si ottiene il seguente messaggio di errore

NullPointerException

Tipi primitivi

I tipi primitivi definiscono tipologie di dati di base, ritenute fondamentali nella stesura di un qualunque programma

Un tipo primitivo non è legato all'esistenza di una classe; i tipi primitivi sono parte integrante del linguaggio di programmazione

Tipi primitivi in Java

I tipi primitivi in Java sono tutti e soli i seguenti:

- *byte* – interi nell'intervallo [-128,127] (8 bit)
- *short* – interi nell'intervallo [-32768, 32767] (16 bit)
- *int* – interi nell'intervallo [-2147483648, 2147483647] (32 bit)
- *long* – interi nell'intervallo $[-2^{63}, 2^{63}-1]$ (64 bit)
- *float* – razionali tra 10^{-45} e 10^{+38} (approx.) in valore assoluto (32 bit)
- *double* – razionali tra 10^{-324} e 10^{+308} (approx.) in valore assoluto (64 bit)
- *char* – caratteri dell'alfabeto Unicode 2.0 (16 bit)
- *boolean* – un valore tra i due seguenti {*true, false*} (8 bit)

Tipi primitivi float e double

I numeri rappresentati dai tipi *float* e *double* si chiamano anche numeri in virgola mobile (floating point)

Questo perché la loro rappresentazione è quella che si usa nelle notazioni scientifiche;
es. 2.3×10^{-5} denota il numero *0.000023*:

- la rappresentazione in virgola mobile consente di memorizzare grandi numeri con pochi bit
- la rappresentazione consiste in una mantissa ed un esponente (nel numero 2.3×10^{-5} la mantissa è *2.3* e l'esponente è *-5*)

Variabili primitive

E’ possibile dichiarare variabili che contengono valori di un qualche tipo primitivo; tali variabili si chiamano variabili primitive

Ad esempio, la dichiarazione

int numero

definisce una nuova variabile primitiva di nome *numero* e di tipo *int*

- *numero* può contenere solo valori nell’intervallo del tipo *int*

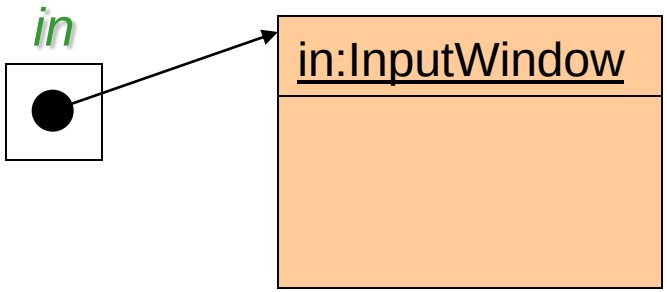
Variabili primitive e riferimento

C’è una differenza sostanziale tra le variabili riferimento e le variabili primitive

- una variabile riferimento memorizza il riferimento ad un oggetto, non direttamente le sue informazioni
- una variabile primitiva memorizza direttamente un dato primitivo

Variabili primitive e riferimento

Ad esempio, se scrivo *InputWindow in = new InputWindow ();* la variabile *in* conterrà il solo riferimento ad un oggetto *InputWindow*



se invece scrivo *int numero = 10;* allora la variabile *numero* conterrà direttamente il valore 10

Letterali primitivi

I tipi primitivi non hanno metodi come le classi, ma possono formare *espressioni*

I più semplici tipi di espressione sono le espressioni costanti, dette anche letterali

Letterali interi

Un letterale *int* è qualsiasi costante compresa nell'intervallo di definizione del tipo *int*:
es. *10, -4, 23, 34000, ….*

Un letterale *long* è qualsiasi costante compresa nell'intervallo di definizione del tipo *long*, seguita da una *'l"* o una *'L"*: es. *10L, -4L, 340000000000L, ….*

Un letterale *int* viene considerato automaticamente un letterale *byte* o *short* qualora assegnato ad una variabile *byte* o *short* che lo può effettivamente contenere

Letterali in virgola mobile

Una costante numerica con virgola è sempre considerata un letterale *double*

es. *10.0, -40.32, 0.002, …..*

Si può anche scrivere *10E0, -4032E-2, 2.0E-3*

Attenzione: *10* è un letterale *int*, mentre *10.0* è *double*

Per indicare i letterali *float*, occorre mettere una *'f'* o una *'F'* alla fine della costante

es. *10.0f, -40.32F, 0.002F, …*

Letterali char e boolean

I letterali *char* sono i singoli caratteri dell'alfabeto Unicode 2.0 (un alfabeto di simboli molto esteso), rappresentati tra apici singoli

es. *'a', 'A', '!', '[', ' ', '1', '\n', '\t', …*

I letterali *boolean* sono i due valori costanti *true* e *false*

Operatori per i tipi primitivi

Oltre ai letterali, anche le singole variabili primitive rappresentano espressioni elementari

E' possibile definire delle espressioni più complesse legando variabili e/o letterali con appositi simboli del linguaggio, chiamati operatori

- ci concentriamo dapprima sull'uso degli operatori applicabili ai numeri interi
- vedremo poi gli operatori per i numeri in virgola mobile e quindi quelli per i char e per i boolean

Operatori base per gli interi

I principali operatori Java per i numeri interi sono gli operatori aritmetici classici: +, -, *, /, dove il simbolo * e / indicano rispettivamente il prodotto e la divisione fra interi

- le regole di precedenza sono le stesse utilizzate nell'algebra
- si possono usare le parentesi tonde per forzare le precedenze
- le parentesi tonde possono avere un qualunque livello di annidamento

Esempi di espressioni tra interi

Alcune espressioni valide composte di soli letterali e relativi risultati:

• *20 – 3 + 1*	*18*
• *4 * 3 + 1*	*13*
• *-4 * (3 + 1)*	*-16*
• *11 / 2*	*5*
• *(1 + ((2+3)*4 + (5-1)/3))*2*	*44*

Espressioni valide composte anche da variabili intere (es. a,b,..):

- *10 * a – b*
- *(b-4)/2 + a*a*
- *(a+a+b) * b/a*

L'operatore resto

Un ulteriore operatore aritmetico tra interi molto utilizzato è l'operatore resto, denotato con *%*.

- se *x* ed *y* sono variabili o letterali interi, *x%y* restituisce il resto della divisione intera tra *x* ed *y*
- in pratica deve sempre risultare che l'espressione *y*(x/y) + x%y* ha lo stesso valore di *x*

Esempi di uso di espressioni con l'operatore resto e relativo risultato

- *-11 % 5* *-1*
- *11 % -5* *1*
- *-11 % -5* *-1*

Assegnazione ed espressioni

Abbiamo già visto l'operatore di assegnazione "=", ma occorre fare alcune precisazioni

Consideriamo il seguente frammento di codice:

```
int a = 2;
a = a + 1;
```

Quanto vale *a* al termine del frammento di codice?

Assegnazione ed espressioni

Una operazione di assegnazione si compone sempre di due parti, quella a sinistra di "=" e quella a destra di "="

- la parte a sinistra deve sempre essere una variabile;
- la parte a destra può essere una qualunque espressione;
- Java valuta prima il valore dell'espressione a destra di "=", e poi assegna il valore di tale espressione alla variabile a sinistra di "="
- il tipo di dato del valore associato all'epressione deve essere compatibile con quello della variabile

Assegnazione ed espressioni

Possiamo ora ritornare sul frammento di codice e rispondere alla domana

```
int a = 2;
a = a + 1;
```

Al termine di questa assegnazione, *a* vale *2*

Nel momento in cui si inizia a valutare l'espressione *a+1*, *a* vale *2*, e quindi *a+1* varrà *3*; il valore *3* è dunque assegnato ad *a*, sovrascrivendo quello precedente

Operatori di assegnazione composta

Le assegnazioni del tipo *a = a+1* sono molto frequenti nei programmi, perché servono ad incrementare (o decrementare) il valore di una variabile

Per snellire i codici dei programmi, Java permette di usare degli operatori di assegnazione composta

a **+=** *x*	equivale a scrivere *a = a+x*
a **-=** *x*	equivale a scrivere *a = a-x*
a ***=** *x*	equivale a scrivere *a = a*x*
a **/=** *x*	equivale a scrivere *a = a/x*
a **%=** *x*	equivale a scrivere *a = a%x*

Operatori di incremento e decremento

Esistono anche due ulteriori operatori, detti operatore di incremento (++) ed operatore di decremento (--)

a++	equivale a scrivere *a = a+1*
a--	equivale a scrivere *a = a-1*

La forma *a*++ è detta *suffissa*, ma è anche possibile usare una forma *prefissa*, cioè ++*a*

Se gli operatori ++ e -- sono usati in espressioni composte, le regole di precedenza variano in funzione della forma (prefissa o suffissa) utilizzata

Operatori di incremento e decremento

Esempi di uso di ++ e -- in espressioni composte:

```
a = 3;
b = 2 + a++;
```

Al termine di questa linea di codice, *a* vale *4* e *b* vale *5*

```
a = 3;
b = 2 + ++a;
```

Al termine di questa linea di codice, *a* vale *4* e *b* vale *6*

Nella notazione suffissa, gli operatori ++ e -- vengono applicati solo dopo la valutazione dell'espressione; in quella prefissa vengono valutati prima (con precedenza rispetto agli altri operatori)

Operatori per i numeri con virgola

Per i numeri in virgola mobile valgono ancora gli operatori aritmetici *+, -, *, /, %*

Quando applicato ai tipi *double* e *float*, l'operatore */* calcola la divisione esatta (senza troncamenti)

11.0 / 5.0 vale *2.2*

Invece *x%y* restituisce il resto (con virgola) rispetto al numero di volte per il quale *y* entra interamente in *x – ad esempio 5.0/2.4 vale 0.2, mentre 5.0/2.5 vale 0*

Operatori per numeri misti

Cosa accade se applico un operatore aritmetico tra numeri interi e numeri con virgola?

La regola in Java dice che il risultato è con virgola se almeno uno degli operandi è con virgola

11/5 vale *2*

11.0/5 vale *2.2*

11/5.0 vale *2.2*

Overflow

Supponi che *a* e *b* siano due variabili *int* tali che *a* vale *2147483647* (valore *int* massimo) e *b* vale *2*; quanto vale la variabile *c* dopo la seguente assegnazione?

int c = a + b;

Il valore dell'espressione *a+b* dovrebbe essere *2147483649* , ma tale numero non può essere rappresentato da un tipo *int*; si parla allora di overflow

Nella variabile *c* verrebbe memorizzato il valore negativo *-2147483647*

Assegnazioni tra tipi diversi

Cosa succede se vogliamo assegnare un valore di un tipo x ad una variabile di un diverso tipo y?

- in generale la cosa non è consentita
- ci sono dei casi in cui ciò è permesso

Il seguente frammento di codice

```
int a = 10;
long b = a;
```

è accettato, poiché l'insieme di definizione del tipo *int* è contenuto propriamente in quello del tipo *long*

Conversione implicita

Se x ed y sono tipi primitivi tali che l'insieme di definizione di y contiene quello di x, allora:

- si può assegnare un valore di tipo x ad una variabile di tipo y;
- il valore assegnato viene automaticamente convertito al tipo y (conversione implicita)

Assegnazioni tra tipi diversi

Il seguente frammento di codice

```
int a = 10;
short b = a;
```

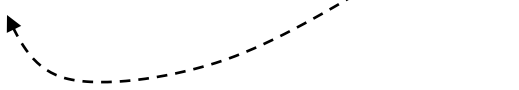
non è accettato, poiché nel convertire un *int* in uno *short* si potrebbe avere una perdita di precisione *(loss of precision)*

Il compilatore non valuta il valore che si vuole assegnare, ma accetta o meno l'assegnazione solo sulla base dei tipi a sinistra e a destra dell'operatore di assegnazione

Conversione esplicita

Quando si è certi che un valore di un certo tipo può essere contenuto in una variabile di un tipo differente, si può ricorrere ad una operazione di conversione esplicita (cast)

```
int a = 10;
short b = (short)a;
```

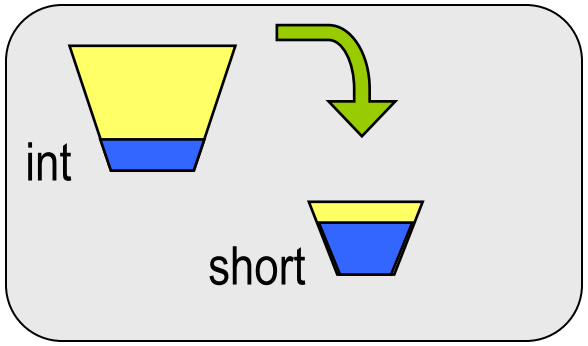


Il codice sopra scritto verrebbe accettato dal compilatore, ma il programmatore si sta assumendo il rischio di una eventuale perdita di precisione

Perdita di precisione

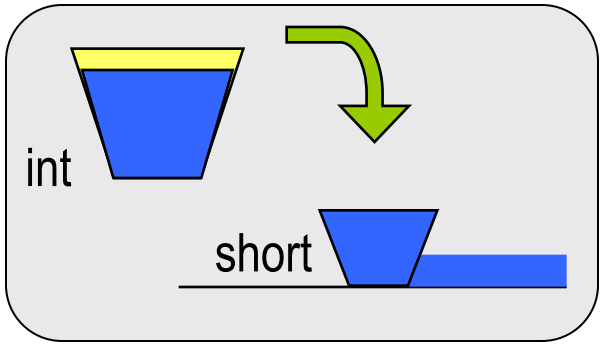
Il codice seguente non causa perdite di precisione

```
int a = 10;
short b = (short)a;
```



Il codice seguente invece causa una perdita di precisione; il valore in *b* sarà *-31072*

```
int a = 100000;
short b = (short)a;
```



Operatori per i char

Ad un tipo *char* si possono applicare tutti gli operatori visti per i numeri interi

- per capire come agiscono questi operatori dobbiamo però fare alcuni chiarimenti sul tipo *char*

L'alfabeto Unicode 2.0 (insieme di definizione dei *char*) permette di rappresentare 2^{16} simboli differenti

- tali simboli sono enumerati, cioè sono messi in corrispondenza con i primi 2^{16} numeri naturali
- ogni simbolo è associato ad un numero che corrisponde alla sua posizione nell'alfabeto

Esempi di posizioni per i char

Ad esempio:

- i simboli ‘a’, ‘b’, ‘c’, ….., ‘z’ (cioè le lettere minuscole dell’alfabeto inglese), si trovano nelle posizioni dalla 97 alla 122;
- i simboli ‘A’, ‘B’, ‘C’, …., ‘Z’ (lettere maiuscole), si trovano nelle posizioni dalla 65 alla 90.
- i simboli ‘0’, ‘1’, ‘2’, ….., ‘9’, si trovano nelle posizioni dalla 48 alla 57

Espressioni con i char

Gli operatori tra *char* agiscono sulle posizioni intere ad essi associate, ed il risultato associato è un *int*

```
char a = 'A';
char b = '2';
char c = (char)(a+b);
```

- 'A' è in posizione 65
- '2' è in posizione 50
- il carattere assegnato alla variabile c sarà quello in posizione 115, cioè 's'

Espressioni di tipo boolean

Il tipo di dato *boolean* è forse il più utilizzato nella programmazione

- Il suo nome è in onore del matematico britannico George Boole (1815-1864)
- il tipo *boolean* è utilizzato per associare un valore ai predicati, cioè espressioni che possono soltanto essere vere (*true*) o false (*false*)
  - esempio di predicato in lingua italiana:
    "10 è maggiore di 2" (true)

Operatori logici

Gli operatori logici sono operatori applicabili a valori *boolean*; il risultato di un operatore logico è ancora un *boolean*

Supponi che *a* e *b* siano valori boolean. Gli operatori logici in Java sono i seguenti:

- l'operatore unario *!* (negazione o NOT)

  !a vale *true* se *a* è *false,* e *false* se *a* è *true*

- l'operatore binario *&&* (congiunzione condizionale o AND)

  a && b vale *true* se e solo se sia *a* che *b* sono *true*

- l'operatore binario || (disgiunzione condizionale o OR)

  a || b vale *true* se e solo se almeno uno tra *a* e *b* è *true*

Operatori logici

E' importante notare che gli operatori *&&* e *||* valutano il secondo operando solo se necessario

- *a && b* valuta *b* solo se *a* è *true*
- *a || b* valuta *b* solo se *a* è *false*

In Java esistono anche operatori logici simili a *&&* e *||*, denotati con *&* e *|*; essi si applicano più in generale a numeri interi per fare i confronti bit a bit; non li studiamo in questo corso

Operatori relazionali

Esistono in Java altri operatori molto usati, che servono a confrontare valori numerici o caratteri, ma il cui risultato è un *boolean*; tali operatori sono noti come operatori relazionali

- l'operatore **maggiore di**, **>**
- l'operatore **maggiore di o uguale a**, **>=**
- l'operatore **minore di**, **<**
- l'operatore **minore di o uguale a**, **<=**
- l'operatore **uguale a**, **==**
  - da non confondere con l'operatore di assegnazione **=** !!!!
- l'operatore **non uguale a**, **!=**

Espressioni relazionali (esempi)

10 > 5	vale *true*
10.0 < 5.0	vale *false*
10 > 10	vale *false*
10 >= 10	vale *true*
10.0 >= 5.0	vale *true*
10 == 5	vale *false*
10 == 10	vale *true*
10 != 5	vale *true*
10 != 10	vale *false*
'a' > 'z'	vale *false*
'a' > 'Z'	vale *true*
'A' > '5'	vale *true*

Predicati complessi

Le espressioni logiche e quelle relazionali possono essere combinate a formare predicati (espressioni di tipo *boolean*) anche molto complessi; ecco alcuni esempi, dove *a, b, c* sono variabili *boolean*, mentre *numero* è di tipo *int*

- *(a && b) || !c*
- *(numero >= 0) && (numero < 10)*
- *((numero < -1) && (numero < -10)) || !(a && c)*

Predicati complessi - esempio

Quanto vale la variabile *d* alla fine di questo frammento di codice? Perché?

```
boolean a, b, c, d;
a = true;
b = true;
c = a && !b;
b = (b && c) || (!b && !c);
a = (b && c) || (!b && !c);
c = !(a && b);
d = (a && b && !c) || (a && !b);
```

Predicati complessi - esempio

Svolgimento

```
boolean a, b, c, d;
a = true;
b = true;
c = a && !b;                              //false
b = (b && c) || (!b && !c);               //false
a = (b && c) || (!b && !c);               //true
c = !(a && b);                            //true
d = (a && b && !c) || (a && !b);          //true
```

Costanti

In Java, oltre alle variabili, è possibile dichiarare delle costanti

Le costanti sono contenitori di dati, come le variabili - anche una costante ha un tipo ed un nome

Il valore di una costante tuttavia non può variare nel tempo; la sua inizializzazione deve avvenire contestualmente alla sua dichiarazione

Dichiarazione di costanti

Una costante va dichiarata mettendo la parola chiave *final* davanti al tipo

- per convenzione, i nomi delle costanti hanno solo caratteri maiuscoli

Ecco alcuni esempi di dichiarazione di costanti:

- *final int A = 10000;*
- *final double PI = 3.141593;*
- *final short PICCOLONUMERO = 10;*

Uso di costanti

L'uso di costanti è raccomandato quando in un programma si deve ricorrere spesso a valori che:

- non debbono cambiare durante l'esecuzione;
- potrebbero cambiare in versioni successive del programma

```
final double PI = 3.14;
double r = ...          // raggio di un cerchio
double area = PI*r*r;
double perimetro = 2*PI*r;
```

Se voglio raffinare il valore di PI, devo cambiare il codice solo in un punto del programma

Glossario dei termini principali

Termine	Significato
Tipi di dato	Tipologia di dato definibile attraverso un linguaggio di programmazione
Tipi riferimento	Tipi di dato associati alla definizione di una classe; una variabile riferimento memorizza il riferimento ad un oggetto di una classe
Tipi primitivi	Tipi di dato elementari che sono parte integrante del linguaggio; In Java: int, double, short, byte, long, float, double, char, boolean.
null	Letterale di tipo riferimento; si può assegnare ad ogni variabile riferimento e rappresenta un riferimento nullo
NullPointerException	Errore generato da un programma quando si tenta di invocare un metodo su un riferimento null
Variabile primitiva	Una variabile di tipo primitivo, cioè che può contenere dati di un qualche tipo primitivo
Letterali	Valori costanti di un qualche tipo di dato
Operatori	Simboli del linguaggio di programmazione che servono ad operare tra tipi di dato (es. operatori aritmetici, operatori logici, operatori relazionali, ecc.)
Overflow	Problema che si verifica quando si tenta di assegnare ad una variabile un valore al di fuori del suo insieme di definizione
Conversione implicita	Operazione di conversione automatica di tipo esercitata dall'interprete Java in fase di esecuzione dei programmi
Conversione esplicita (cast)	Operazione di conversione di tipo comandata dal programmatore

Glossario dei termini principali

Termine	Significato
Predicati	Espressioni di tipo boolean, cioè il cui valore è di tipo boolean
Costante	Contenitore di dati che non può cambiare il suo valore nell'escuzione di un programma (il valore va assegnato nella dichiarazione)
final	Parola chiave del linguaggio Java che va messa davanti alla definizione di una costante