Lab. Programmazione (CdL Informatica) & Informatica (CdL Matematica) a.a. 2022-23

Monica Nesi

Università degli Studi dell'Aquila

19 Ottobre 2022

Espressioni e Operatori

Per poter manipolare i dati e poter esprimere proprietà su di essi, abbiamo bisogno di *operatori* per formare *espressioni* di un dato tipo, a partire da valori costanti e variabili dello stesso tipo.

Consideriamo le *espressioni aritmetiche* e le *espressioni booleane* in Java e gli operatori corrispondenti.

Le espressioni aritmetiche sono formate applicando gli operatori aritmetici usuali a partire da costanti, variabili e altre espressioni di tipo aritmetico.

Lo stesso discorso vale per le espressioni booleane (o logiche).

Operatori aritmetici

Oltre agli usuali operatori aritmetici, abbiamo i seguenti operatori in Java (andando da quelli di massima precedenza a quelli di minima precedenza):

- - (op. unario), ++ (autoincremento), -- (autodecremento);
- * (prodotto), / (divisione), % (modulo);
- + (somma), - (differenza);
- = (assegnamento).

Gli operatori su una stessa riga hanno la stessa precedenza.

Gli operatori +, - (binario), *, / e % sono *associativi a sinistra*.

L'operatore di assegnamento = *associa a destra*.

La precedenza può essere modificata tramite l'introduzione di parentesi.

Operatori aritmetici: esempi

L'operatore di modulo % restituisce il *resto* della divisione intera fra l'operando di sinistra e quello di destra.

Ad esempio, dati `a = 23` e `b = 6`, si ha che:

```
int q = a/b, r = a%b;
```

risulta in `q = 3` ed `r = 5`.

Date le dichiarazioni con assegnamenti:

```
int a = 4 * 5 + 2;
int b = a = 3;
```

dopo il primo assegnamento il valore di `a` è 22, mentre l'operazione di *assegnamento multiplo* assegna 3 prima alla variabile `a` e poi alla variabile `b`.

Operatori ++ e --

Gli operatori ++ e -- permettono di incrementare/decrementare di una unità il valore della variabile a cui sono applicati.

Data una variabile intera `x`, tali operatori possono essere applicati *prima* della variabile (prefissi), i.e. `++x` e `--x`, oppure *dopo* la variabile (postfissi), i.e. `x++` e `x--`.

N.B. Se sono usati come operatori (e non come comandi di incremento/decremento), la semantica nei due casi è diversa!

Se l'operatore ++ (--) *precede* la variabile, allora *prima* viene incrementato (decrementato) il valore della variabile e *poi* viene valutata l'intera espressione.

Se l'operatore ++ (--) *segue* la variabile, allora *prima* viene valutata l'intera espressione e *poi* viene incrementato (decrementato) il valore della variabile.

Operatori ++ e --: esempi

Dato il frammento di codice:

```
int a = 4, b = 3;
int c = ++a + b;
```

si ha `a = 5`, `b = 3` e `c = 8`.

Mentre nel caso seguente:

```
int a = 4, b = 3;
int c = a++ + b;
```

si ha `a = 5`, `b = 3` e `c = 7`.

Gli operatori `++`, `--` e `=` hanno *effetti collaterali*.

Meglio usarli come *comandi* (che modificano lo stato della macchina, i.e. il valore delle variabili) piuttosto che come operatori all'interno di espressioni.

Operatori relazionali

Confrontare valori numerici risulta in un valore booleano, i.e. `true` oppure `false`.

Gli operatori *relazionali* (o di confronto) sono i seguenti (in ordine decrescente di precedenza):

- > (maggiore di), < (minore di), >= (maggiore o uguale), <= (minore o uguale);
- == (uguale a), != (diverso da).

N.B. L'operatore relazionale di uguaglianza == è diverso dall'operatore di assegnamento =.

Operatori relazionali: esempi

Dato il frammento di codice:

```
int a = 4, b = 3;
boolean p = (a++ + b) == (++a + b);
```

si ha `a = 6`, `b = 3` e `p = false`.

Mentre nel caso seguente:

```
int a = 4, b = 3;
boolean p = (--a + b) != (a-- + b);
```

si ha `a = 2`, `b = 3` e `p = false`.

Operatori logici o booleani

Le espressioni logiche (o booleane) sono formate applicando gli operatori logici a partire dalle costanti `true` e `false`, da variabili di tipo `boolean` e da altre espressioni di tipo `boolean`.

Noi considereremo solo tre operatori logici basilari: *negazione* (NOT) (op. unario), *congiunzione* (AND) e *disgiunzione* (OR) (entrambi op. binari).

Nel seguito vediamo la *sintassi* Java per questi operatori (i.e. i simboli che sono usati nel linguaggio per denotare tali operazioni) e la loro *semantica* tramite le *tabelle di verità*.

Il valore di verità di un'espressione logica dipende dai valori di verità delle sue sotto-espressioni.

Assumiamo che p e q denotino espressioni logiche.

Negazione

Sintassi Java: !
Semantica: se p è vera, allora !p è falsa.
Se p è falsa, allora !p è vera.

Tabella di verità per la negazione

p	!p
`true`	`false`
`false`	`true`

Esempi:
`!true` vale `false`.
`!false` vale `true`.
`!(!p)` ha lo stesso valore di verità di `p`.

Congiunzione

Sintassi Java: &
Semantica: Se p e q sono vere, allora $p\&q$ è vera.
Altrimenti $p\&q$ è falsa.

Tabella di verità per la congiunzione

p	q	$p\&q$
`true`	`true`	`true`
`true`	`false`	`false`
`false`	`true`	`false`
`false`	`false`	`false`

Disgiunzione

Sintassi Java: |
Semantica: Se p è vera oppure se q è vera, allora $p|q$ è vera.
Altrimenti $p|q$ è falsa.

Tabella di verità per la disgiunzione

p	q	$p\|q$
`true`	`true`	`true`
`true`	`false`	`true`
`false`	`true`	`true`
`false`	`false`	`false`

La semantica è equivalente a:
Se p e q sono false, allora $p|q$ è falsa. Altrimenti $p|q$ è vera.

Congiunzione e Disgiunzione: esempi e proprietà

Congiunzione:
`true&p` ha lo stesso valore di `p`.
`false&p` vale `false`.
`p&p` ha lo stesso valore di `p`.
`p&!p` vale `false`.
`p&q` ha lo stesso valore di `q&p`.

Disgiunzione:
`false|p` ha lo stesso valore di `p`.
`true|p` vale `true`.
`p|p` ha lo stesso valore di `p`.
`p|!p` vale `true`.
`p|q` ha lo stesso valore di `q|p`.

Varianti *short circuit* && e ||

In Java esistono varianti, dette *short circuit*, per congiunzione e disgiunzione, denotate con && e || rispettivamente.

N.B. La semantica di && e || è la *stessa* di & e |, ma l'implementazione è diversa.

Ciò si basa sul fatto che spesso è possibile sapere il valore della congiunzione/disgiunzione di due valori solo in base al primo valore.

Data la congiunzione di due espressioni p e q, se il valore della prima espressione p è falso, allora la congiunzione è falsa, *indipendentemente* dal valore di q, che quindi non viene valutata.

Solo se il valore di p è vero, allora si valuta q ed inoltre il valore di q sarà il valore della congiunzione di p e q.

Varianti *short circuit* && e || (cont.)

Data la disgiunzione di due espressioni `p` e `q`, se il valore della prima espressione `p` è vera, allora la disgiunzione è vera, *indipendentemente* dal valore di `q`, che quindi non viene valutata.

Solo se il valore di `p` è falso, allora si valuta `q` ed inoltre il valore di `q` sarà il valore della disgiunzione di `p` e `q`.

Valutazione degli operatori logici (binari):

strict per `&` e `|`
lazy per `&&` e `||`

Precedenze degli operatori logici

Gli operatori logici visti hanno le seguenti precedenze (in ordine decrescente):

- `!` (NOT);
- `&` (AND);
- `|` (OR);
- `&&` (AND short circuit);
- `||` (OR short circuit).

Nei codici Java presentati nel corso verranno utilizzate le varianti `&&` e `||`.

Espressione condizionata

In Java esiste l'operatore ternario `?:` che permette di scrivere *espressioni condizionate*, ovvero espressioni il cui valore dipende (i.e. è condizionato) dal valore di un'espressione booleana.

Esempio: l'espressione

```
x == y ? 1 : -1
```

vale `1` se il valore di `x` è uguale a quello di `y`, altrimenti vale `-1`.

La sintassi dell'espressione condizionata è:

$$<BEspr> \ ? \ <Espr> : <Espr>$$

dove $<BEspr>$ denota l'insieme delle espressioni booleane e $<Espr>$ denota l'insieme delle espressioni del linguaggio.

Espressione condizionata (cont.)

```
int n = x == y ? 1 : -1;
```

Alla variabile `n` di tipo `int` viene assegnato `1` o `-1` a seconda del valore dell'espressione booleana `x == y`.

L'operatore `?:` ha precedenza sull'operatore di assegnamento `=`, quindi si valuta prima l'espressione condizionata.

Tale operatore non è strettamente necessario (al suo posto si può usare il comando condizionale `if`, come vedremo).

Si usa se una maggiore concisione migliora la leggibilità del codice.

Massimo tra `x` ed `y`:

```
x > y ? x : y
```

Valore assoluto di `x`:

```
x >= 0 ? x : -x
```

Precedenze operatori: riepilogo

Tutti gli operatori visti in ordine decrescente di precedenza:

- ()
- !, - (unario), ++, --
- *, /, %
- +, -
- >, <, >=, <=
- ==, !=
- &
- |
- &&
- ||
- ?:
- =