Operatori in Java

Programmazione
Corso di laurea in Informatica

Operatori in Java

- Operatori aritmetici
- Operatori di uguaglianza e relazionali
- Operatori logici in espressioni booleane
- Operatori di incremento e decremento
- Operatori di assegnamento
- Operatore condizionale
- La precedenza degli operatori
- L'associatività degli operatori

Espressioni aritmetiche

- Gli operatori aritmetici vengono usati in espressioni aritmetiche e operano con operandi di tipo numerico:
  - `byte`, `short`, `int`, `long`, `float`, `double`
- Il tipo del valore di ritorno calcolato dall'espressione dipende dal tipo dei suoi operandi
  - Se tra gli operandi almeno uno è di tipo virgola mobile allora il risultato è di tipo virgola mobile

Espressioni booleane

- Espressioni in cui compaiono gli *operatori* di Java *di uguaglianza* o *relazionali*, che riportano valori **booleani**

`==`	**uguale**
`!=`	**non uguale**
`<`	**minore**
`>`	**maggiore**
`<=`	**minore o uguale**
`>=`	**maggiore o uguale**

- Si noti la differenza tra l'operatore di uguaglianza (`==`) e l'operatore di assegnamento (`=`)
- Vengono usate principalmente per esprimere le condizioni in istruzioni di controllo del flusso

Il tipo boolean

- Un valore `boolean` rappresenta una condizione di verità o falsità
- Una variabile di tipo boolean può rappresentare un valore a due stati
  - come un interruttore che è ***acceso*** o ***spento***
- Le parole riservate (lettrerali costanti) `true` e `false` sono gli unici valori ammessi per il tipo boolean

```
boolean eseguito = false;
```


Operatori logici

- Nelle espressioni booleane si possono usare gli ***operatori logici***

`!`	`not`
`&&`	`and`
`||`	`or`

- che richiedono operandi di tipo *boolean* e producono un risultato *boolean*
- L'operatore logico `not` è un operatore unario (ha un solo operando)
- Gli operatori logici `and` e `or` sono operatori binari (richiedono due operandi)

Operatore logico `not`

- L'operatore logico **`NOT`** è anche chiamato *negazione logica* o *complemento logico*
- Se una condizione booleana **`a`** è vera, allora **`!a`** è falsa; se **`a`** è falsa, allora **`!a`** è vera
- Le espressioni logiche usano quindi la *tabella di verità* che segue

a	!a
true	false
false	true

Gli operatori logici `and` e `or`

- L'espressione logica **`and`**

  `a && b`

  è vera se entrambi gli operandi **`a`** e **`b`** sono veri, ed è falsa altrimenti

- L'espressione logica **`or`**

  `a || b`

  è vera se **`a`** o **`b`** o entrambi sono veri, ed è falsa altrimenti

Tavole di verità

- Una tavola di verità mostra le possibili combinazioni di termini di valori vero/falso
- Poiché **`&&`** e **`||`** hanno due operandi ciascuno, ci sono 4 possibili combinazioni

a	b	a && b	a || b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Gli operatori logici

- Gli operatori logici vengono usati per formare espressioni booleane complesse
  - Ad es condizioni in istruzioni di selezione o cicli

```
if (totale < MAX && !trovato)
   System.out.println ("Processing…");
```

- Gli operatori logici hanno relazioni di precedenza tra loro e con altri operatori

Espressioni booleane

- Gli operatori logici e di relazione possono essere combinati per ottenere espressioni booleane complesse
- Attenzione, da errore:

```
if ( 0 < numero < 1000) …
if ( car == 'a' || 'b') …
```

- Occorre scrivere:

```
if ( 0 < numero && numero < 1000)
if (car == 'a' || car == 'b')
```


Metodi predicativi

- Un *metodo predicativo* restituisce un valore di tipo **`boolean`**:

```
public class ContoBancario {
   public boolean e'Scoperto() {
        return this.saldo() < 0
   }
}
```

- Esempi predefiniti nella classe Character

  `isDigit`, **`isLetter`**, **`isUpperCase`**

Variabili booelane

- Qualunque variabile che può assumere solo due valori può essere dichiarata di tipo boolean

```
private boolean coniugato;
if (coniugato) …
```

e non

```
if (coniugato == true) …
```

- Si chiamano anche *flag*

Tavole di verità

- Le espressioni vengono valutate usando le tavole di verità

```
(totale < MAX && !trovato)
```

totale < MAX	trovato	!trovato	totale < MAX && !trovato
false	false	true	false
false	true	false	false
true	false	true	true
true	true	false	false

Legge di De Morgan

- Espressioni complesse come:

```
if (!(0<numero && numero<1000))
```

non è vero che 0 < numero e numero < 1000

possono essere semplificate per essere rese più leggibili usando la legge di De Morgan (1806-1871)

- `!(a && b)` equivale a `!a || !b`
- `!(a || b)` equivale a `!a && !b`

Semplificazione con De Morgan

- L'espressione

```
if (!(0<numero && numero<1000))
```

si semplifica

```
if (!(0<numero)||!(numero<1000))
```

e ancora

```
if ((0>=numero)||(numero>=1000))
if ((numero<=0)||(numero>=1000))
```


Operatori di incremento e decremento

- Gli operatori di incremento e decremento sono operatori aritmetici unari
- L'operatore di *incremento* (`++`) aggiunge 1 al suo operando
- L'operatore di *decremento* (`--`) sottrae 1 al suo operando
- L'istruzione

```
cont++;
```

equivale all'istruzione

```
cont = cont + 1;
```


Operatori di incremento e decremento

- Operatori di incremento e decremento possono essere usati in *forma prefissa* (prima della variabile) o in *forma postfissa* (dopo la variabile)
- Quando si usano soli in una istruzione, le due forme sono equivalenti.

`cont++;` equivale a `++cont;`

Operatori di incremento e decremento

- In un'espressione, le due forme possono avere effetti molto diversi
- Sempre la variabile viene aumentata o decrementata
- Ma il valore usato nell'espressione dipende dalla forma prefissa o postfissa:

espressione	operazione sulla variabile	valore usato nell'espressione
`cont++`	somma 1	precedente
`++cont`	somma 1	nuovo
`cont--`	sottrae 1	precedente
`--cont`	sottrae 1	nuovo

Operatori di incremento e decremento

- se `cont` contiene attualmente il valore 45, allora

```
totale = cont++;
```

assegna 45 a `totale` e 46 a `cont`

- se `cont` contiene attualmente il valore 45, allora

```
totale = ++cont;
```

assegna il valore 46 sia a `totale` sia a `cont`

Operatori di assegnamento

- L'operando di destra di un operatore di assegnamento può essere un'espressione
- L'espressione di destra viene dapprima valutata quindi il risultato viene assegnato alla variabile, il cui precedente valore viene sovrascritto
- Nell'istruzione

```
risultato /= (totale-MIN) % num;
```

si calcola prima il valore dell'espressione

```
((totale-MIN) % num);
```

quindi si valuta `risultato / valore_espressione` e lo si assegna a `risultato`

Operatori di assegnamento

- Spesso eseguiamo operazioni di aggiornamento del valore di una variabile utilizzando sempre alcune operazioni (somma o sottrazione …)
- Alcuni operatori di *assegnamento* consentono questo processo
- Esempio:

```
num += cont;
```

equivale a

```
num = num + cont;
```


Operatori di assegnamento

operatore	esempio	equivale a
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

Operatore condizionale

- L'operatore *condizionale* valuta una condizione booleana che determina quale espressione, tra due possibili, valutare
- Il risultato dell'espressione selezionata diventa il risultato dell'operatore condizionale

```
condizione ? Espressione_1 : espressione_2
```

- Se `condizione` è vera, allora viene valutata `espressione_1` altrimenti si valuta `espressione_2`

Operatore condizionale

- L'operatore condizionale è simile all'istruzione if-else, tranne che *riporta il valore di un'espressione*

```
maggiore = (num1 > num2) ? num1 : num2;
```

se `num1` è maggiore di `num2`, allora a `maggiore` viene assegnato `num1` altrimenti `num2`

- L'operatore condizionale è un operatore *ternario*, cioé richiede tre operandi

Operatore condizionale

```
System.out.println
  ("Il resto è di " + cont +
     (cont == 1) ? "lira" : "lire");
```

- se `cont` è 1, allora si stampa "`lira`". Per qualunque altro valore di `cont`, si stampa "`lire`"

Precedenza operatori

- Gli operatori Java hanno una precedenza assegnata, che occorre conoscere
  - Tabella precedenza operatori Java
- Tra gli operatori di massima precedenza ci sono le `()` per la valutazione dei parametri, l'operatore `dot` e gli operatori di auto-incr o auto-decr
- Tra quelli di minima ci sono gli assegnamenti

Associatività degli operatori

- La maggior parte degli operatori binari sono associativi a sinistra
  - Vengono valutati da sinistra a destra

    `a + b + c + d + e` equivale a

    `(a + b) + c) + d) + e)` e non a

    `a + (b + (c + (d + e)))`
- L'operatore di assegnamento è associativo a destra
  - Viene valutato da destra a sinistra