

Programmazione

Corso di Laurea in Informatica

Corso di Laurea in Informatica per la Comunicazione Digitale

Università di Camerino

Prof. Michele Loreti

Anno Accademico 2025/26

Programmazione

Assegnamento

- Il comando di assegnamento
- Scambiare il contenuto di due variabili
- Differenza tra comando ed espressione
- Espressioni impure
- Assegnamento e uguaglianza a confronto
- Incremento e decremento di variabili
- Esempi di incremento e decremento
- Esercizi



Il comando di assegnamento



```
r = 1 + 2 * 3;
```

Sintassi

```
identificatore = espressione ;
```

- `identificatore` è la variabile di cui si vuole modificare il contenuto
- `espressione` produce il valore che si intende scrivere nella locazione di memoria riservata per la variabile

Significato (da raffinare in seguito)

1. Valuta `espressione`
2. Sovrascrivi il contenuto di `identificatore` con il valore di `espressione`

L'assegnamento è **distruttivo**, nel senso che il valore precedentemente contenuto nella variabile viene *sovrascritto* con il valore dell'espressione

Scambiare il contenuto di due variabili

Sbagliato

```
int a = 1, b = 2;          // a == 1, b == 2
a = b;                    // a == 2, b == 2
b = a;                    // a == 2, b == 2
```

- il primo assegnamento sovrascrive il “vecchio” valore di a
- il secondo assegnamento non ha alcun effetto

Corretto

```
int a = 1, b = 2;          // a == 1, b == 2
int t = a;                // a == 1, b == 2, t == 1
a = b;                    // a == 2, b == 2, t == 1
b = t;                    // a == 2, b == 1, t == 1
```

- si usa una **variabile temporanea** per memorizzare il “vecchio” valore di a

Differenza tra comando ed espressione



Le parti di un programma Java si possono dividere (grossolanamente) in due categorie dette **comandi** ed **espressioni**

Un **comando**, quando viene **eseguito**, modifica lo **stato** del calcolatore

- stampa di un messaggio (`println`)
- lettura di un dato (`StdIn.readInt`)
- assegnamento di una variabile (`a = b`)

Una **espressione**, quando viene **valutata**, produce un **valore**

- `1 + 2 * 3 ~~> 7`
- `"abc" + "def" ~~> "abcdef"`
- `"A = " + a ~~> "A = 1"`

Espressioni impure



In Java esistono **espressioni impure**, ovvero espressioni la cui valutazione non solo produce un risultato, ma ha anche dei cosiddetti **effetti collaterali** che modificano lo stato del calcolatore

Il comando di assegnamento

```
identificatore = espressione ;
```

è anche un'espressione il cui valore coincide con quello di espressione

```
int a = 1;
println("A = " + a);      // stampa A = 1
println("A = " + (a = 2)); // stampa A = 2
println("A = " + a);      // stampa A = 2
```

Uso tipico: assegnamento multiplo

```
a = b = c = 1;
```

Nota: l'operatore di assegnamento = è **associativo a destra**

Assegnamento e uguaglianza a confronto

$a = a + 1$ (assegnamento)

1. Valuta l'espressione $a + 1$ usando il valore corrente di a
2. *Modifica* a sovrascrivendola con il valore ottenuto (espressione impura)
3. Produce il valore ottenuto come risultato

$a == a + 1$ (uguaglianza)

1. Confronta il valore corrente di a con il valore corrente di a sommato a 1
2. *Non modifica* a (espressione pura)
3. Produce sempre false come risultato

Tranelli

```
if (a = true) comando // modifica a ed esegue sempre comando
if (a = false) comando // modifica a e non esegue mai comando
```

Morale: preferire condizione a condizione == true e !condizione a condizione == false

Incremento e decremento di variabili



```
x = x + 1; // incrementa x di 1  
y = y - 1; // decrementa y di 1
```

Incremento e decremento di variabili sono *particolarmente frequenti* in programmi Java e per questo motivo esistono *operatori unari specifici*

Sintassi	Operatore	Effetto	Valore prodotto
<code>++x</code>	pre incremento	incrementa x di 1	x dopo l'incremento
<code>x++</code>	post incremento	incrementa x di 1	x prima dell'incremento
<code>--y</code>	pre decremento	decrementa y di 1	y dopo il decremento
<code>y--</code>	post decremento	decrementa y di 1	y prima del decremento

Note

- le espressioni che fanno uso di `++` e `--` sono *impure*
- `x = x + 1` è equivalente a `++x`, **non** a `x++`
- `y = y - 1` è equivalente a `--y`, **non** a `y--`

Esempi di incremento e decremento

```
int x = 1, y = 5;          // x == 1, y == 5
println(x + " , " + y);   // x == 1, y == 5, stampa 1, 5
x++;
// x == 2, y == 5
y--;
// x == 2, y == 4
println((x++) + " , " + y); // x == 3, y == 4, stampa 2, 4
println((++x) + " , " + y); // x == 4, y == 4, stampa 4, 4
println(x + " , " + (y--)); // x == 4, y == 3, stampa 4, 4
println(x + " , " + (--y)); // x == 4, y == 2, stampa 4, 2
++x;
// x == 5, y == 2
--y;
// x == 5, y == 1
println(x + " , " + y);   // x == 5, y == 1, stampa 5, 1
```

Nota: è buona norma evitare l'uso di ++ e -- in espressioni complesse, perché ne rendono difficile la comprensione

```
int x = 3;
println("Pessima idea " + (double) (++x) / (--x));
```

- come minimo ci sono due esiti possibili (1.333 e 1.5) in base all'ordine di valutazione delle espressioni (da sinistra a destra, da destra a sinistra)

Esercizi



1. Descrivere il valore prodotto e l'effetto della seguente espressioni impura, supponendo che `a` sia una variabile di tipo `int`

```
a = a++
```

2. Se `a` è una variabile di tipo `int`, per quali valori di `a` le condizioni `a++ > 0` e `++a > 0` sono vere?
3. Scrivere un programma che legge un numero intero $0 \leq x < 64$ e determina il numero di bit che occorrono per rappresentare x . Il programma deve stampare “fuori scala” se x non è compreso nell’intervallo indicato.
4. Scrivere un programma che legge un numero intero $0 \leq x < 64$ e determina il numero di bit a 1 nella rappresentazione binaria di x , stampando “fuori scala” se x non è compreso nell’intervallo indicato.
5. Scrivere un programma che legge una sequenza di tre numeri interi (memorizzati in tre variabili distinte) e conta il numero di **inversioni** della sequenza, ovvero quante volte un numero letto **prima** è più grande di un numero letto **dopo**.