

# Operatori su Java

Al livello più basso i dati in Java sono **manipolati tramite operatori**, la sintassi è simile al C o C++.

## Print Statement


La sintassi della stampa sullo schermo è la seguente:

```
System.out.println("Un sacco da scrivere");
```

 Esistono delle librerie che ti permettono di semplificarne la scrittura.

## Operatori

Un operatore ha il compito di **prendere uno o più argomenti in e produrre un nuovo valore**. Tutti gli operatori producono un valore a partire dai loro operandi:

 La maggior parte degli operatori funziona solo con variabili primitive, le eccezioni sono:

1. '=', '==' e '!=' che funzionano per tutti gli oggetti.
2. '+' e '+=' che funzionano con la classe String

Un qualcosa di importante è conoscere l'ordine di precedenza delle diverse operazioni, conviene sempre utilizzare le parentesi:

```
public class precedenza {  
    public static void main(String[] args) {  
        int x = 1, y = 2, z = 3;  
        int a = x + y - 2/2 + z; // a = 5, prima divisione  
        int b = x + (y - 2)/(2 + z); // b = 1, prima parentesi  
        System.out.println("a = " + a + " b = " + b);  
    }  
}
```

✍ In questa situazione abbiamo un '+' anche nel println che sta a significare la concatenazione di stringhe, praticamente converte la non stringa in una stringa.

## Assegnazione

Utilizziamo l'operatore '=', l'assegnazione in ambito di numeri primitivi è abbastanza veloce.

Si prende il valore a destra e lo si assegna alla variabile a sinistra:

```
int a = 4;
int b = a;
```

In questa situazione *a* ha il valore 4, *b* copia il valore di *a*, cioè 4, se proviamo a modificare *a* il valore *b* rimarrà lo stesso.

La situazione cambia quando cerchiamo di manipolare un oggetto, infatti ciò che stiamo facendo è copiare una **reference**, quindi:

```
class Tank{
    int level; //classe tank con variabile intera
}

public class Assignment{
    public static void main(String[] args)
    {
        Tank t1 = new Tank();
        Tank t2 = new Tank();

        t1.level = 9;
        t2.level = 47;

        System.out.println("1: t1.level: " + t1.level + ", t2.level: " +
t2.level);

        t1 = t2; //uguaglianza tra oggetti, adesso gli oggetti puntano alla
stessa reference;

        //t1 punta allo stesso oggetto di t2

        System.out.println("2: t1.level: " + t1.level + ", t2.level: " +
t2.level);

        t1.level = 27; //se cambio valore a t1 cambia anche quello di t2

        System.out.println("3: t1.level: " + t1.level + ", t2.level: " +
```

```

t2.level);

        //Se avessimo fatto t1.level = t2.level allora avremo ancora oggetti
        indipendenti.

    }

}

```

Quindi se faccio *oggetto1 = oggetto2* sto puntando il primo oggetto al secondo oggetto di conseguenza se cambio il primo cambio anche il secondo e viceversa.

Questo fenomeno di chiama **Aliasing**, avviene anche nelle **chiamate di funzione**

## Operazioni matematiche

Funzionano alla stessa maniera della maggior parte dei linguaggi di programmazione. Quindi abbiamo:

- Addizione
- Sottrazioni
- Divisioni
- Moltiplicazioni
- Moduli

```

import java.util.*;
public class operazioni {

    public static void main(String[] args)
    {
        //Create a seeded random number generator
        Random rand = new Random(47);
        int i,j,k;

        //Sceglie un valore da 1 a 100
        j = rand.nextInt(100)+1;
        System.out.println("j : " + j);

        i = rand.nextInt(100)+1;
        System.out.println("i: "+i);

        k = rand.nextInt(100)+1;
    }
}

```

```

System.out.println("k: "+k);

//Somma fra due numeri
i = j+k;
System.out.println("j + k: " + i);

//Differenza fra due numeri
i = j-k;
System.out.println("j - k: " + i);

//Divisione fra due numeri
i = j/k;
System.out.println("j / k: " + i);

//Moltiplicazione fra due numeri
i = j*k;
System.out.println("j * k: " +i);

//Modulo fra due numeri
i = j%k;
System.out.println("j % k: "+ i);

//Da notare come queste sono assegnazioni e non aliasing perché stiamo
usando delle variabili

//Possiamo fare questi test anche con i float e double
float u, v;

u = rand.nextFloat();
v = rand.nextFloat();

float w = u + v;

System.out.println("u + v: " + w);

//Queste operazioni funzionano anche per char,
//byte, short, int, long, double

}

}'''

```

>[!note] **Classe Random:**

> **Java** ci permette di generare numeri casuali con l'oggetto **\*\*Random\*\*** che prende in input un **==seme==**, questo va a specificare il tipo di numeri casuali che si ottengono, se non si passa l'argomento allora saranno completamente

casuali perchè utilizzerebbe come metodo l'orario attuale.

> Inoltre la classe `Random` ==chiama anche dei metodi== che servono a generare diversi tipi di numeri casuali come: `**nextInt()`, `nextFloat**`.

Vale anche scrivere:

```
``` java
int a = 4;
a++;
a--;
```

Per aggiungere o togliere un valore dall'elemento *a*.

## Operatori relazionali e logici

Generano un valore booleano (vero o falso) e servono per **comparare diversi valori**:

- **Maggiore, minore, uguale, ecc...**

I valori logici sono i classici (**&&**) AND, (**||**) OR e (**!**)NOT producono anche questi un valore booleano:

```
public class operatorirelazionali {

    public static void main(String[] args)
    {
        int a = 10;
        int b = 20;
        int c = 10;
        int d = 20;
        boolean e = true;
        boolean f = false;

        System.out.println("a == b: " + (a == b));
        System.out.println("a != b: " + (a != b));
        System.out.println("a > b: " + (a > b));
        System.out.println("a < b: " + (a < b));
        System.out.println("b >= a: " + (b >= a));
        System.out.println("b <= a: " + (b <= a));
        System.out.println("a == c: " + (a == c));
        System.out.println("b != d: " + (b != d));

        System.out.println("e AND f: " + (e && f));
        System.out.println("e OR f: " + (e || f));
        System.out.println("NOT f: " + (!f));
```

```
}  
  
}
```

Quando utilizziamo gli operatori logici potremmo imbatterci nello **Short Circuiting**, alcuni parti della equazione logica non vengono calcolate, perché si sa già quale è il risultato finale.

#### Esempio:

Immaginiamo di avere tre metodi chiamati test1(), test2() e test3() e facciamo questa operazione: test1() AND test2() AND test3().

Se ottenessimo che test1() è un valore false allora sarebbe inutile svolgere gli altri due test perché il valore alla fine sarà neccessariamente falso.

## Operatori bitwise e shift

Java permette di eseguire delle **operazioni sui bit di numeri interi**, di seguito tutti gli operatori:

- **&**: (AND su bit);
- **|**: (OR su bit);
- **^**: (XOR su bit);

Le variabili booleane sono trattate come valori ad 1 bit.

Un'altra maniera per manipolare i valori binari è tramite gli **operatori di shifting**, possono essere usati solo su tipi interi e sono:

- **>**: sposta di uno a destra tutte le cifre binarie;
- **<**: sposta di uno a sinistra tutte le cifre binarie;

```
public class bitwise {  
  
    public static void main(String[] args) {  
  
        int a = 10101010;  
        int b = 10101011;  
  
        System.out.println("a & b = " + (a & b)); //and su bit  
        System.out.println("a | b = " + (a | b)); //or su bit  
        System.out.println("a ^ b = " + (a ^ b)); //xor su bit  
        System.out.println("~a = " + ~a); //negazione su bit
```

```

    int c = 5;

    System.out.println(Integer.toBinaryString(c)); //stampa in binario il
valore 5 in binario 101

    c = c << 2; //shift a sinistra di 2 bit del valore 101

    System.out.println(Integer.toBinaryString(c)); //stampa in binario il
valore 10100

    System.out.println(c); //stampa il valore corrispettivo al binario
10100 = 20

    char d = 'A';
    System.out.println(Integer.toBinaryString(d)); //stampa in binario il
valore 65 = 1000001

    //65 perche' il char viene convertito in int e il valore
corrispondente e' 65
}

}

```

## Operazioni di Casting

Il Casting è molto semplice in Java, infatti il tipo di una **variabile verrà cambiato nel momento più opportuno alla situazione.**

Se diamo ad un intero un valore float, Java convertirà subito l'intero in un float.

**Tutti i tipi primitivi permettono il casting tranne i booleani e nemmeno le classi lo permettono.**