



La sintassi JAVA

- **Tipi di dati, variabili e costanti**
- **Operatori matematici, logici e di confronto**
- **Controllo del flusso: istruzioni condizionali e loop**

[Tipi di dati]

- **Java è un linguaggio fortemente tipizzato**
 - Necessario dichiarare le variabili
- **Otto tipi di dati primitivi**
 - Sei di tipo numerico
 - Quattro interi e due a virgola mobile
 - Uno utilizzato per la memorizzazione di caratteri
 - Uno utilizzato per la memorizzazione di valori veri o falsi
- **In Java tutti i tipi numerici hanno dimensione fissata**
 - Indipendenza dalla piattaforma

Tipi di dati numerici

Tipo	Requisiti di memorizzazione	Intervallo
byte	1 byte	da -128 a 127
short	2 byte	da -32768 a 32767
int	4 byte	da -2147483648 a 2147483647
long	8 byte	da -9 miliardi di miliardi a 9 miliardi di miliardi
float	4 byte	6-7 cifre decimali significative
double	8 byte	15 cifre decimali significative

[Altri tipi di dati primitivi]

■ char

- In Java i caratteri non sono numeri
 - Necessario un cast esplicito

■ boolean

- In C/C++ non esiste il tipo boolean
 - Valore diverso da zero denota vero, mentre uguale a zero denota falso
- In Java non è possibile convertire un valore booleano in un valore numerico
- Grazie al tipo boolean si evita il classico errore C/C++
 - Assegnazione invece di uguaglianza in costrutti che prevedono una condizione

[Variabili - 1]

■ Dichiarazione di variabili

- Tutte le variabili prima di essere utilizzate devono essere dichiarate
 - tipo della variabile (identificativo standard)
 - nome variabile (identificativo scelto dall'utente)
- Per convenzione il nome delle variabili inizia con un carattere minuscolo

■ Inizializzazione

- Spesso alla variabile deve esser assegnato un valore iniziale
- Dichiarazione e inizializzazione possono essere fatte nella stessa istruzione

[Variabili - 2]

- **Java permette la dichiarazione e l'inizializzazione delle variabili in qualunque parte del codice**
 - Dichiarazione della variabile al momento del bisogno
 - Evitare di seguire questo approccio
 - Dichiarare le variabili all'inizio del metodo aumenta la leggibilità, la manutenzione e la qualità del codice
- **C/C++ distinguono dichiarazione e definizione**
 - In Java non esistono dichiarazioni separate dalle definizioni

[Variabili di riferimento]

■ Variabile

- Identificatore a cui si può attribuire un valore
 - “si supponga che x valga 5”
 - “posto y pari al valore della temperatura esterna ...”

■ Variabile di riferimento (reference variable)

- Una variabile il cui valore è un riferimento ad un oggetto
- Dichiarazione simile alle variabili
 - classe
 - identificatore

[Conversioni tra tipi numerici]

■ Conversione implicita

- Se gli operandi non sono dello stesso tipo vengono convertiti al tipo più “grande”
- E’ sempre possibile assegnare una variabile di un tipo ad una di un tipo “più grande”

■ Conversione esplicita

- Consiste nell’indicare il tipo di destinazione tra parentesi, seguita dal nome della variabile

[Costanti]

- **Utilizzo della parola chiave final**
 - Si assegna il valore una sola volta (rimane impostato per sempre)
 - Generalmente una variabile **final** si definisce anche **static**
 - Si parla di costanti di classe
 - Per convenzione i nomi delle costanti sono indicati con caratteri maiuscoli
- **const pur essendo una parola riservata non è usata per definire costanti**

[Operatori matematici]

Simbolo	Operazione
+	Addizione / Concatenazione
-	Sottrazione
*	Moltiplicazione
/	Divisione
%	Divisione con modulo (solo per interi)
++	Incremento unitario
--	Decremento unitario

Utilizzo degli operatori

■ Operatori di incremento e decremento

- `x++;` /* equivale a scrivere `x = x + 1;` */
- `x--;` /* equivale a scrivere `x = x - 1;` */

■ Possono trovarsi in due posizioni

- Post-fissi `x++`
 - Il valore viene aggiornato “dopo” la valutazione dell’espressione in cui si trova
- Pre-fissi `++x`
 - il valore viene aggiornato “prima” della valutazione

Forma contratta degli operatori

■ Operatori aritmetici

```
espressione1 = espressione1 <operatore> espressione2  
/* questa risulta equivalente alla sottostante */  
espressione1 <operatore> = espressione2
```

■ Esempio

- **x = x + 3;** **/* diventa */**
- **x += 3;**
- **y = y * 5;** **/* diventa */**
- **y *= 5;**

Operatori di confronto

Simbolo	Operazione	Utilizzo
==	Uguale a	$a == b$
!=	Diverso da	$a != b$
<	Minore di	$a < b$
>	Maggiore di	$a > b$
>=	Maggiore o uguale a	$a >= b$
<=	Minore o uguale a	$a <= b$

Operatori logici

Simbolo	Operazione	Utilizzo
&&	AND logico	(a && b)
	OR logico	(a b)
!	NOT logico	!(a && b)

■ Applicazione della “*valutazione cortocircuitata*”

- (a && b)
 - Se a è falsa allora b non viene valutata
- (a || b)
 - Se a è vera allora b non viene valutata

Precedenza degli operatori - 1

- **Si applicano le regole solite**
 - Moltiplicazione, divisione e modulo prima di addizione e sottrazione e concatenamento
 - Gli operatori che hanno la stessa precedenza sono valutati da sinistra a destra
- **L'utilizzo delle parentesi può cambiare tali precedenze**
 - Usare le parentesi rende le cose più chiare
 - E' fortemente consigliato utilizzare sempre le parentesi

Precedenza degli operatori - 2

- Ordine di valutazione dell'espressione:

$$a + b + c + d + e$$

1 2 3 4

$$a + b * c - d / e$$

3 1 4 2

$$a / (b + c) - d \% e$$

2 1 4 3

$$a / (b * (c + (d - e)))$$

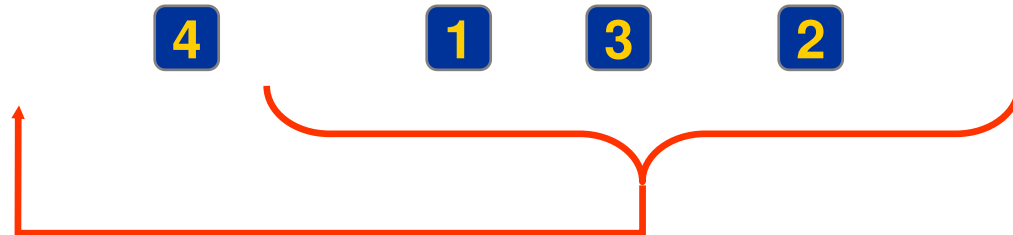
4 3 2 1

[Operatore di assegnamento]

- Ha la precedenza più bassa di qualunque altro operatore

Prima si valuta l'espressione alla destra dell'operatore =

```
risposta = somma / 4 + MAX * altro;
```



Poi il risultato è assegnato alla variabile alla sinistra

[**Classi wrapper - 1**]

- **I tipi di dati primitivi non sono oggetti, e quindi non hanno metodi**
- **La realizzazione di operazioni complesse, quali valore assoluto o elevamento non sono offerte quali operatori dal linguaggio Java**
- **E' necessario definire dei metodi opportuni**
 - In quale classe?
 - Quali oggetti bisogna invocare?

[Classi wrapper - 2]

■ Oggetti da invocare

- Metodi statici, nessun oggetto da invocare
- Parametri rappresentati da tipi primitivi

■ Classi

- Esistono svariate classi predefinite che servono a raggruppare metodi statici logicamente correlati
- Classe **Math** – metodi statici orientati alla matematica
 - Es: `static int abs(int a)`
- Classe **Integer**, **Double**, **Long**, ecc – metodi statici per i relativi tipi primitivi

[Controllo del flusso]

■ Controllo del flusso in JAVA

- Istruzioni condizionali
- Loop

■ Blocchi di istruzioni

- Istruzioni semplici circonscritte da una coppia di parentesi graffe
- Possono essere annidati
- Definiscono l'ambito delle variabili
 - Non è possibile definire variabili di nome identico in blocchi annidati

Istruzioni condizionali - 1

- Esecuzione selettiva di codice a seconda della falsità o verità di una condizione
- L'istruzione **if**

```
if (condizione)
    istruzione o blocco // true action
else
    istruzione o blocco // false action
```

- Le alternative sono mutuamente esclusive
 - else non è obbligatorio
- Operatore ?

```
(condizione) ? e1 : e2
```

Istruzioni condizionali - 2

■ Verifiche a più vie (cascaded if-else)

```
if (condizione1)
    istruzione o blocco
else if (condizione2)
    istruzione o blocco
else if (condizione3)
    istruzione o blocco
.....
else
    istruzione o blocco
```

- else è sempre riferita all'istruzione if più vicina

Istruzioni condizionali - 3

■ Costrutto switch

- if/else poco per gestire condizioni con molte alternative
 - E' possibile valutare solo char e numeri (tranne long)

```
switch (variabileDaTestare) {  
    case valore1:  
        .....  
        break;  
  
    .....  
    case valore2:  
        .....  
        break;  
    default:  
        .....  
        break;  
}
```

■ Break etichettato

Loop indeterminati

■ Il costrutto while

```
while (condizione)
    istruzione o blocco
```

- Esegue il corpo del loop finché la condizione è vera
 - La verifica della condizione avviene all'inizio del ciclo

■ Il costrutto do-while

```
do {
    // Corpo
} while (condizione)
```

- Simile al while, ma verifica la condizione alla fine

Loop determinati

■ Il costrutto for

```
for (inizializzazione; condizione; incremento)  
  istruzione o blocco
```

- Esegue il corpo del loop finché la condizione è vera
 - La verifica della condizione avviene all'inizio del ciclo
- Il contatore è spesso dichiarato all'atto dell'inizializzazione
 - Visibilità limitata all'interno del loop
- Attenzione alla verifica dell'uguaglianza dei numeri in virgola mobile

[Loop: equivalenza]

- Il costrutto for è equivalente al costrutto while

```
for (inizializzazione; condizione; incremento) {  
    // Corpo del ciclo  
}
```

```
inizializzazione;  
while (condizione) {  
    // Corpo del ciclo  
    incremento;  
}
```

[Array: introduzione]

■ In Java gli array sono oggetti

- Un array è un riferimento ad un oggetto di tipo array
- Non è facile ridefinire la dimensione di un array
- E' facile accedere agli elementi
- Si creano attraverso la parola chiave new

```
int[] arrayOfInt = new int[10];
```

■ Creazione e inizializzazione di un array

```
int[] arrayOfInt = { 1, 3, 5, 7, 9 }
```

- Possibilità di avere array multidimensionali

[Array: copia]

- Un variabile array è un riferimento a un oggetto

```
int[] arrayOfInt1 = { 1, 3, 5, 7, 9 }  
int[] arrayOfInt2 = arrayOfInt1;  
  
arrayOfInt2[1] = 5;  
    // Modifica anche arrayOfInt1
```

- Necessario usare il metodo arraycopy della classe System

```
System.arraycopy(from, fromIndex, to, toIndex, count);
```

[La classe Arrays]

- **Fornisce dei metodi statici per lavorare con gli array**
 - `void sort(XXX[] a)`
 - Ordina a adottando l'algoritmo QuickSort
 - Il tipo degli elementi dell'array è primitivo
 - `int binarySearch(XXX[] a, XXX v)`
 - Cerca v in a utilizzando l'algoritmo di ricerca binaria
 - Se v è presente viene restituito il suo indice, altrimenti viene restituito un valore negativo r ($-r + 1$ la posizione corretta di v)
 - `void fill(XXX[] a, XXX v)`
 - Imposta tutti gli elementi di a su v