Parte seconda: espressioni, operandi e operatori.

1. Le espressioni

Le **espressioni** sono combinazioni di operatori e operandi. Gli operandi sono a loro volta combinazioni di costanti, variabili semplici e strutturate, chiamate metodi.
Un'espressione in Java rappresenta qualunque combinazione valida di tali parti.

2. Gli operandi

Variabili locali e globali: dichiarazione e tipo
Prendono il nome di **variabili locali** le variabili utilizzate all'interno dei metodi. Le variabili locali sono deallocate al termine dell'esecuzione del metodo; tali variabili hanno, quindi, lo stesso tempo di vita del metodo in cui sono definite.
La visibilità delle variabili locali è solo all'interno del metodo in cui sono definite. Java non possiede variabili globali, in altre parole variabili visibili da tutte le parti del programma.
Poiché Java è un linguaggio orientato agli oggetti, bisogna ragionare in termini di oggetti e di interazione tra oggetti, anziché in termini di funzioni e sottoprogrammi.
Per dichiarare una variabile bisogna specificarne il tipo. Ad esempio, per creare una variabile di tipo stringa:

```
String nome;
```

Variabili del medesimo tipo possono essere dichiarate sulla stessa riga.

```
String nome;
String cognome;
```
equivale a
```
String nome, cognome;
```

L'inizializzazione della variabile può avvenire sulla stessa riga della dichiarazione.

```
int altezza = 180;
```

I tipi primitivi
La tabella che segue rappresenta i tipi di dato primitivi:

TIPO	DESCRIZIONE	CAMPO DI VARIABILITÀ
`int`	Intero normale	32 bit (da –2147483648 a +2147483647) Per riferirsi a questi valori estremi si utilizzano le costanti `Integer.MIN_VALUE` e `Integer.MAX_VALUE`
`short`	Intero corto	16 bit (da –32768 a +32767) Per riferirsi a questi valori estremi si utilizzano le costanti `Short.MIN_VALUE` e `Short.MAX_VALUE`
`long`	Intero lungo	64 bit (da –9223372036854775808 a +9223372036854775807) Per riferirsi a questi valori estremi si utilizzano le costanti `Long.MIN_VALUE` e `Long.MAX_VALUE`
`byte`	Intero byte	8 bit (da –128 a +127)
`boolean`	Booleano	true – false
`char`	Carattere	16 bit senza segno (da 0 a 65536)
`float`	Numeri in virgola mobile a precisione singola	32 bit (da 1.4E-45 a ~3.4E38) Per riferirsi a questi valori estremi si utilizzano le costanti `Float.MIN_VALUE` e `Float.MAX_VALUE`
`double`	Numeri in virgola mobile a precisione doppia	64 bit (da 4.9E-324 a ~1.8E308) Per riferirsi a questi valori estremi si utilizzano le costanti `Double.MIN_VALUE` e `Double.MAX_VALUE`

Costanti
Le costanti (o valori letterali) sono quantità esplicite, e quindi non vanno dichiarate. Le costanti servono a fornire informazioni a espressioni o attributi e metodi. Nel linguaggio Java, per dichiarare il valore di una costante, vengono utilizzati gli stessi tipi usati per le variabili. Vengono quindi usati tipi:

numerici
- **interi** (positivi e negativi)
- **decimali** (in virgola mobile. In notazione scientifica o standard)
- **booleani** (true - false)

carattere singolo
stringhe, che comprendono
- valore nullo
- caratteri speciali

1. **Costanti numeriche**: le costanti decimali si compongono di due parti: parte intera e parte decimale. Le due parti vengono separate dal punto, secondo la notazione anglosassone. Ad esempio:

   ```
   double altezza = 179.50;
   ```

   Le costanti in virgola mobile producono numeri in virgola mobile di tipo double, qualunque sia la precisione.
   Aggiungendo il suffisso f (oppure F) al valore numerico, si ottiene una costante di tipo float:

   ```
   float altezza = 179.50f;
   ```

   Come già accennato, per le costanti in virgola mobile si può anche impiegare la notazione esponenziale, costituita da mantissa ed esponente, dove l'esponente può anche essere negativo.

   `1.7E100` oppure `0.5E-3`

2. **Caratteri**: le costanti di tipo carattere sono simboli singoli racchiusi fra apici singoli. I caratteri sono memorizzati come codici Unicode a 16 bit.

3. **Stringhe:** le costanti stringa hanno la seguente forma:

   ```
   "<Testo>"
   ```

   Vengono utilizzati i doppi apici all'inizio e alla fine di un qualsiasi testo. La differenza con le costanti carattere è che le costanti carattere vengono dichiarate con l'apice singolo, mentre le stringhe vengono dichiarate con il doppio apice.

4. **Caratteri speciali:** si supponga di dovere, all'interno di una stringa, andare a capo e di inserire i doppi apici (") per racchiudere una parte di testo. Come fare? In entrambi i casi dobbiamo utilizzare dei caratteri speciali. Un carattere speciale viene indicato facendolo precedere dalla barra rovesciata "\". Il compilatore, in questi casi, interpreta i caratteri in maniera differente. Ad esempio:

   ```
   System.out.println("Prima riga\nSeconda riga");
   ```

   L'output prodotto è:

   ```
   Prima riga
   Seconda riga
   ```

   Esistono quindi dei caratteri speciali non stampabili che vengono espressi mediante una sequenza speciale di caratteri che inizia con \. Questa sequenza prende nome di **sequenza di escape**. Per stampare un qualsiasi carattere Unicode utilizzeremo la seguente notazione:

   ```
   \uNNNN
   ```

   Dove "u" rappresenta Unicode e NNNN indica il valore esadecimale del codice Unicode relativo al carattere desiderato, espresso su quattro cifre. Per rappresentare solamente i primi 128

caratteri Unicode, è possibile utilizzare il valore relativo espresso in ottale del codice Unicode. Per far questo si utilizza la notazione

```
\<val_ott>
```

dove "val_ott" è il valore espresso in ottale del codice Unicode.

3. Gli operatori

La caratteristica essenziale degli operatori è quella che producono sempre un risultato e, alcuni, modificano anche uno degli operandi. Gli operatori vengono espressi o con un singolo carattere, o con due o con tre caratteri, senza spazi di separazione.
Gli operatori di dividono in:

- Operatori di assegnamento
- Operatori aritmetici
- Operatori relazionali
- Operatori logici
- Operatori sui bit
- Operatori su stringhe e oggetti

Operatori di assegnamento: l'operatore di assegnamento in Java è il carattere "=". In Java questo operatore non viene usato come in matematica, poiché serve ad assegnare un valore e non ad effettuare confronti. Per effettuare confronti si utilizza l'operatore "==" (doppio uguale). L'operatore "=" valuta l'espressione a destra ed il risultato viene assegnato alla variabile a sinistra. Ad esempio, supponiamo di dover assegnare valore 10 alla variabile *n*:

```
n = 10;
```

Operatori aritmetici: gli operatori aritmetici si dividono in:

- Operatori unari
- Operatori binari

Operatori unari: gli operatori unari si applicano ad un solo operando e ne modificano il valore. Hanno la forma

```
<operando><operatore>
```
(operatori unari postfissi)

oppure

```
<operatore><operando>
```
(operatori unari prefissi)

Gli operatori unari si distinguono in operatori di:

Incremento: ++ ad esempio `n++`
Decremento: -- ad esempio `n--`

Gli operatori unari aritmetici possono essere prefissi e postfissi. L'operatore prefisso modifica l'operando prima di valutarne il valore, mentre l'operatore postfisso modifica l'operando solo dopo averne valutato il valore. Per esempio:

```
n = 5;
n2 = n++;
```

Risultato: n2 = 5 e n = 6

```
n = 5;
n2 = ++n;
```

Risultato: n2 = 6 e n = 6

Operatori binari: gli operatori binari si applicano a due operandi e non cambiano il loro valore, ma memorizzano unicamente il loro risultato. Gli operatori binari hanno forma:

```
<primo_operando><operatore><secondo_operando>
```

Gli operatori binari utilizzati in Java sono i seguenti:

Operatore	Simbolo	Risultato
Addizione	+	Somma due operandi
Sottrazione	-	Sottrae il secondo operando dal primo
Moltiplicazione	*	Moltiplica i due operandi
Divisione	/	Divide il primo operando per il secondo
Modulo	%	Restituisce il resto della divisione tra due operandi interi

L'operatore di assegnamento può anche essere scritto in forma compatta, cioè abbinato ad un operatore aritmetico. Si utilizza la forma compatta quando si ha una situazione del tipo:

```
x = x + y;
```

Una espressione del genere può anche essere scritta:

```
x += y;
```

Si ha, quindi, la seguente tabella:

Forma completa	Forma compatta
`x = x + y;`	`x += y;`

`x = x - y;`	`x -= y;`
`x = x / y;`	`x /= y;`
`x = x * y;`	`x *= y;`
`x = x % y;`	`x %= y;`

Operatori relazionali: gli operatori relazionali restituiscono un valore logico, vero o falso: vero se la relazione è verificata, falso se non è verificata. Di seguito sono elencati gli operatori relazionali usati in Java:

Operatore	Significato
>	maggiore di
<	minore di
>=	maggiore o uguale
<=	minore o uguale
==	uguale a
!=	diverso da

Operatori logici: gli operatori logici richiedono come operandi delle epressioni booleane e restituiscono un risultato booleano.

Operatore	Simbolo	Significato
`OR`	`||` e `|`	OR logico
`AND`	`&&` e `&`	AND logico
`NOT`	!	Negazione logica
`XOR`	^	OR esclusivo

Segue la tabella di verità degli operatori logici:

X	Y	X || Y	X && Y	X ^ Y	!X
F	F	F	F	F	V
F	V	V	F	V	V
V	F	V	F	V	F
V	V	V	V	F	F

Quindi:

- OR: il risultato ha valore true se almeno uno dei due operandi ha valore true
- AND: il risultato ha valore true se entrambi gli operandi assumono valore true
- XOR: il risultato ha valore true se uno solo dei due operandi ha valore true
- NOT: il risultato ha valore true se l'operando ha valore false (inverte il valore dell'operando)

Operatori sui bit: questi operatori operano sulle rappresentazioni binarie degli operandi e restituiscono come risultato un numero intero. Il valore 1 di un bit corrisponde al valore booleano true, il valore 0 corrisponde al valore booleano false. Gli operatori sui bit sono i seguenti:

Operatore	Descrizione
`&`	AND bit a bit
`|`	OR bit a bit
`^`	XOR bit a bit
`~`	Complemento a uno
`>>`	Shift a destra con segno
`<<`	Shift a sinistra con riempimento di zeri
`>>>`	Shift a destra senza segno

Gli operatori di shift effettuano uno spostamento di tutti i bit dell'operando sinistro verso destra (>> e >>>) e verso sinistra (<<) del numero di posizioni specificate dall'operando destro.

Consideriamo degli esempi:

```
byte b = 100;
b >> 1;
```

La rappresentazione binaria del numero decimale 100 è 01100100. Lo shift verso destra di una posizione produrrà il seguente risultato:

0	1	1	0	0	1	0	0	
0	0	1	1	0	0	1	0	0 (perso)

Il valore che otterremo è 00110010, che corrisponde al numero decimale 50.
Supponiamo di avere una variabile X, pari a:

```
X = 10010011;
```

Consideriamo le espressioni >> e >>>.
L'espressione `X>>>2` fornisce come risultato 00100100. Infatti l'operatore >>> riempie di zeri i due bit iniziali. Se l'espressione fosse stata `X>>2` il risultato ottenuto sarebbe 11100100, perché l'operatore >> propaga il segno nelle prime due posizioni iniziali.
L'operatore di complemento a uno inverte lo stato dei bit, per cui ogni 0 sarà trasformato in 1 e viceversa.

Java consente di eseguire operazioni logiche sui bit. Nelle seguenti tabelle di verità sono riportati i possibili risultati prodotti dall'applicazione degli operatori AND, OR, XOR e complemento a uno. Tutte le combinazioni sono state effettuate considerando un singolo bit degli operandi.

& (AND)		
X	**Y**	**X & Y**
1	1	1
1	0	0
0	1	0
0	0	0
| (OR)		
X	**Y**	**X | Y**
1	1	1
1	0	1
0	1	1
0	0	0

^ (XOR)		
X	**Y**	**X ^ Y**
1	1	0
1	0	1
0	1	1
0	0	0

~ (COMPLEMENTO)	
X	**~X**
1	0
0	1

Operatori su stringhe: gli operatori su stringhe sono gli operatori + e +=. L'operatore + serve per concatenare le stringhe. Ad esempio:

```
String a = "Ciao ";
String b = "mondo";
String c = a + b;
```

Il risultato sarà "Ciao mondo".

L'operatore += accoda le stringhe. Per esempio:

```
String a = "Sono il ";
a += " programma che accoda stringhe";
```

Il risultato sarà "Sono il programma che accoda stringhe".

Avremo modo di conoscere meglio le stringhe (che sono oggetti che fanno parte della classe *String*) più avanti.

Operatori sugli oggetti: gli operatori sugli oggetti sono gli operatori == e `!=`. L'operatore == è l'operatore di uguaglianza. Ad esempio, avendo due stringhe, prima e seconda:

```
prima == seconda
```
(oggetti di classe String)

L'operatore `!=` è l'operatore di disuguaglianza. Ad esempio, sempre considerando il caso di avere due stringhe, prima e seconda:

```
prima != seconda
```
(oggetti di classe String)

In questo esempio abbiamo utilizzato ancora oggetti di classe String, ma gli operatori == e `!=` si applicano ad oggetti di qualsiasi classe.

Operatori polimorfi: un operatore è polimorfo quando opera su operandi di tipo diverso. In Java, gli operatori polimorfi sono:

```
+
+=
&
|
==
!=
```

Per esempio, l'operatore + è un operatore polimorfo, perché opera sia su operandi numerici sia su stringhe.

L'operatore ternario '?': l'operatore '?' è un operatore ternario. La sua sintassi è la seguente:

```
<espressione_1> ? <espressione_2> : <espressione_3>
```

espressione_1 è un'espressione booleana, che viene valutata. Se essa risulta vera viene valutata espressione_2, se risulta falsa viene valutata espressione_3. Come vedremo in seguito, questo operatore ternario viene a volte utilizzato in alternativa al costrutto if..else. Un esempio in cui viene utilizzato '?':

```
int ore = 15;
String saluto = (ore >= 17) ? "Buona sera" : "Buona giornata";
```

Nell'esempio, se la variabile ore è maggiore o uguale a 17, il messaggio di saluto sarà "Buona sera", altrimenti sarà "Buona giornata".

4. Conversioni di tipo

In Java la conversione di tipo può avvenire in due modi: **conversione implicita** e **conversione esplicita**.

Conversione implicita:

Conversione da numerico a stringa: se vengono concatenati operandi di tipo diverso prevale il valore String. Ad esempio:

```
SN = 4 + "ac"
```
risultato:
```
SN = "4ac"
```

Il risultato è di tipo String.

Conversione tra tipi numerici a virgola mobile: se il tipo di arrivo è maggiore del tipo di partenza, avviene una conversione di tipo. In questi casi il tipo di arrivo è più grande e dà maggiore precisione. Nessuna informazione viene persa. Ad esempio:

```
float f = 12.34F;
double d = f;
```

Questo tipo di assegnamento è corretto. Infatti notiamo che il tipo float è minore del tipo double (float è rappresentato a 32 bit, mentre double a 64 bit).

TIPO DI ARRIVO	TIPO DI PARTENZA
`int`	`double, float`
`double`	`long`
`double`	`Byte, float, int`
`float`	`long, int`

Conversione esplicita: la conversione esplicita può avvenire **effettuando un casting** o utilizzando **metodi di conversione**.

Casting esplicito: se il tipo di arrivo è inferiore rispetto al tipo di partenza, è possibile effettuare un casting esplicito. Per effettuare un casting esplicito, la sintassi è la seguente:

```
(<tipo>)<val>
```

Dove 'tipo' è il tipo di arrivo e 'val' è il tipo di partenza. Se, ad esempio, volessimo convertire un double in un int, scriveremmo:

```
double d = 123.45;
int i = (int)d;
```

Il valore di i sarà 123, con la perdita dei decimali. I casting che possono essere effettuati sono riassunti nella seguente tabella:

Metodi per la conversione:

Da stringa a numero: si utilizzano i metodi parseInt e parseDouble rispettivamente delle classi Integer e Double per convertire una stringa in un valore numerico di tipo int o double.

```
String SN = "40";
int i = Integer.parseInt(SN);
double d = Double.parseDouble(SN);
```

Da numero a stringa: si utilizza il metodo valueOf della classe String.

```
int i = 125;
String s = String.valueOf(i);
```

Nella prossima parte vedremo le strutture per il controllo del flusso, gli array e le stringhe.