

SUPSI

# Tipi di Dato Primitivi

Loris Grossi, Fabio Landoni, Andrea Baldassari

Contenuto realizzato in collaborazione con: T. Leidi, A.E. Rizzoli, S. Pedrazzini

Fondamenti di Informatica

Bachelor in Ingegneria Informatica

# Obiettivo

Essere in grado di utilizzare in maniera corretta i tipi di dato primitivi messi a disposizione da Java e i vari operatori.

Obiettivi della lezione:

- Capire l'utilità e le caratteristiche dei tipi di dato.
- Conoscere i tipi numerici integrali e la loro rappresentazione in memoria.
- Conoscere i tipi numerici in virgola mobile e la loro rappresentazione in memoria.
- Conoscere i dati boolean ed i char.
- Conoscere i vari operatori (d'assegnamento, relazionali, logici, aritmetici, d'incremento e di decremento).
- Conoscere le precedenze dei vari operatori.
- Conoscere le conversioni implicite ed esplicite.

## Variabili

Prima di poter essere utilizzata, una variabile deve essere dichiarata. Ogni dichiarazione di variabile, in Java, è composta da un **tipo di dato** e da un **identificatore** (nome).

Quando viene dichiarata, la variabile può anche essere inizializzata.

La variabile è accessibile, tramite l'identificatore, dal momento in cui viene dichiarata e fino alla fine del blocco in cui essa è contenuta.

```
int abc; // non inizializzata
int x = 20 + 30;

int y = 0, z;
y = x * 4;
z = x * 8;

double d = z / (y * 1.75);
String mioNome = "Pippo";
```

## Esempio variabili

```
public class UsoDiVariabili {  
    public static void main(String[] args) {  
        int valore = 0;  
        System.out.println("Valore iniziale: " + valore);  
  
        valore = valore + 1;  
        System.out.println("Valore incrementato: " + valore);  
  
        String str1 = "Buongiorno";  
        String str2 = "studenti";  
  
        double altroValore = valore + 5.316;  
  
        System.out.println(str1 + " " + str2 + " il valore "  
                           + "finale è: " + altroValore);  
    }  
}
```

## Costanti

Per le **costanti**, in Java bisogna utilizzare le **variabili static final** da dichiarare al di fuori (di regola prima) della procedura main.

```
static final int LA_MIA_COSTANTE = 42;
```

## Esempio con costanti

```
public class Sfera {  
  
    static final double PI = 3.141592653589793;  
  
    public static void main(String[] args) {  
        double raggio = 4.;  
        double vol = (4. / 3.) * PI * raggio * raggio * raggio;  
        System.out.println(vol);  
    }  
}
```

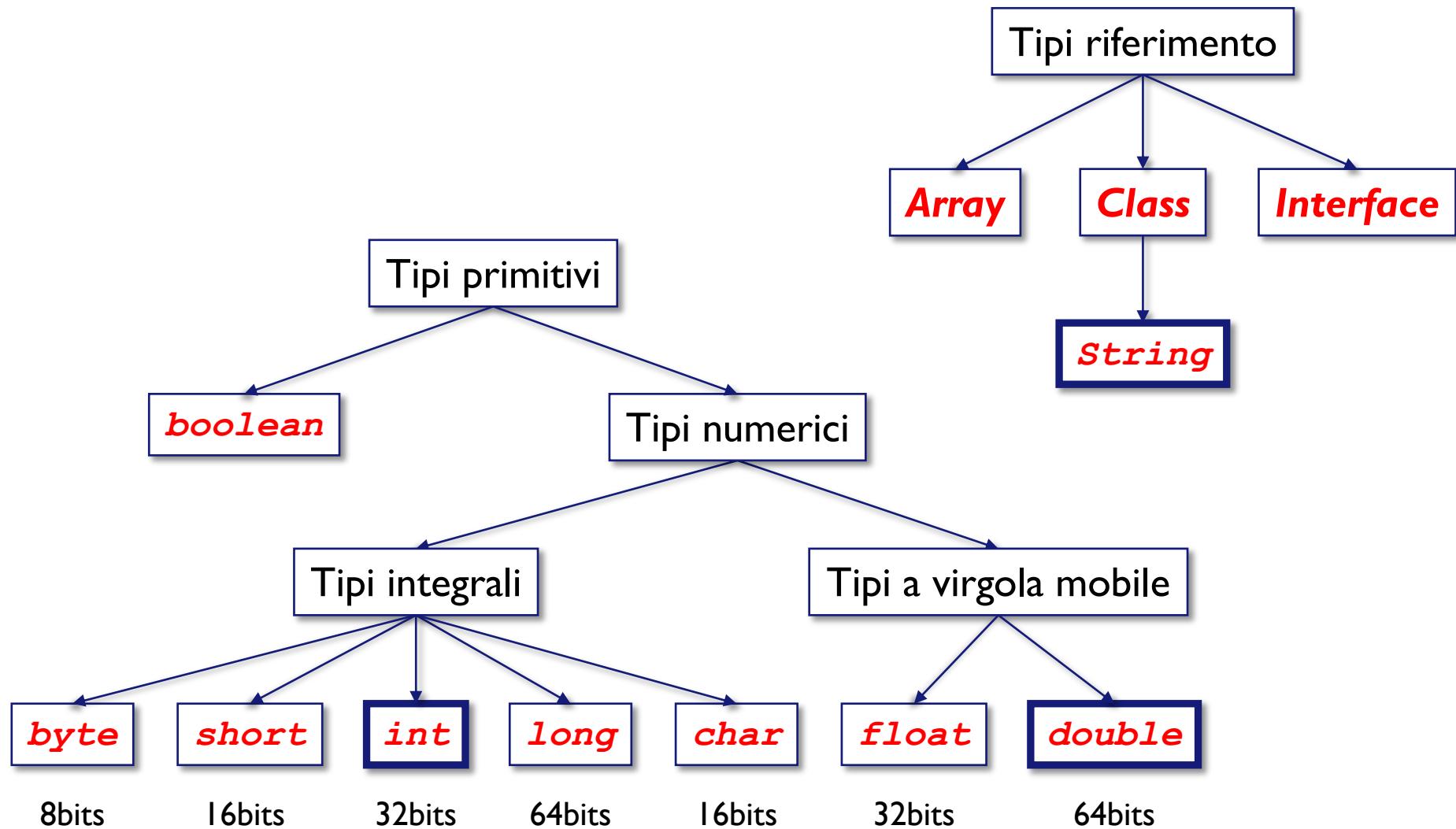
## Tipi di dato

“Il linguaggio di programmazione Java è **strongly typed**. Questo significa che ogni variabile e ogni espressione **ha un tipo conosciuto al momento della compilazione**. I tipi **limitano i valori** che una variabile può assumere o che un’espressione può produrre, **limitano le operazioni** supportate su quei valori e determinano il significato di quelle operazioni. Lo strong typing aiuta ad **individuare gli errori** durante la compilazione.”

— fonte: *Java language specification*

Ripasso

# Tipi di dato

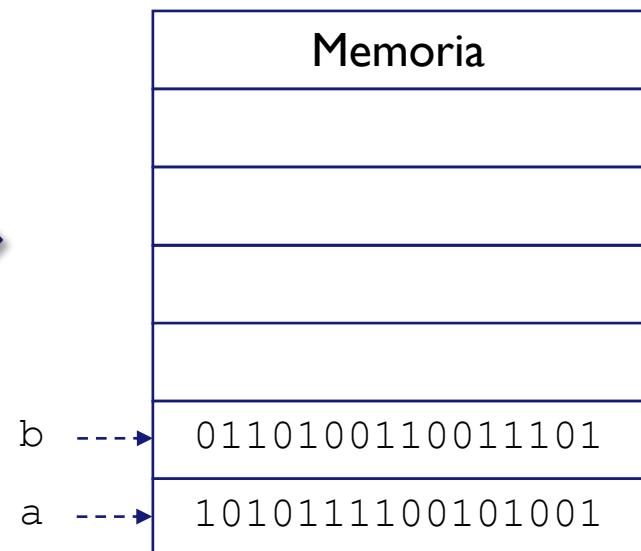
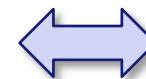


# Utilità dei tipi di dato

I tipi di dati permettono di caratterizzare:

- lo **spazio** riservato ed utilizzato dalle variabili in memoria;
- i **valori** che possono essere assunti dalle variabili di quel tipo;
- le **operazioni** che possono essere effettuate su tali variabili.

```
int a;  
  
// Parte del programma  
  
float b;  
  
// Resto del programma
```



# Tipi di dato primitivi

Caratteristiche principali dei tipi di dato primitivi:

<i>Tipo</i>	<i>Memoria</i>	<i>Range di valori ammessi</i>	<i>Default</i>
boolean	Non specificato	true o false	false
byte	8 bit	Da -128 a 127 inclusi	0
short	16 bit	Da -32'768 a 32'767 inclusi	0
int	32 bit	Da $-2^{31}$ a $2^{31}-1$ inclusi	0
long	64 bit	Da $-2^{63}$ a $2^{63}-1$ inclusi	0L
char	16 bit	Da '\u0000' a '\uffff' cioé da 0 a 65'535 inclusi	'\u0000'
float	32 bit	Single-precision 32-bit IEEE 754 floating point	0.0f
double	64 bit	Double-precision 64-bit IEEE 754 floating point	0.0

## Il tipo boolean

Il tipo **boolean** è un tipo che può assumere esclusivamente 2 valori: **true** e **false**.

I valori booleani si ottengono anche dalla **valutazione di espressioni condizionali**, quali ad esempio:

```
boolean risultato = tasso > 0.05;
```

Questo tipo di dato è utilizzato nell'algebra logica, detta anche algebra booleana. Il nome deriva da **George Boole** che, nel 19esimo secolo, fu il primo a definire un sistema algebrico logico.

## Tipi numerici

In Java, per i tipi di dato numerici, i tipi utilizzati per **default** sono ***int*** (se tipi interi) e ***double*** (se tipi a virgola mobile).

In Java **NON** esistono le versioni ***unsigned*** (senza segno) dei tipi di dato numerici (come esistono, ad esempio, nel linguaggio C).

## Tipi numerici interi: byte, short, int e long

Rappresentano **un'approssimazione di  $\mathbb{Z}$**  (insieme dei numeri interi relativi).

I tipi di dato **byte** e **short** sono simili al tipo di dato `int` ma occupano meno spazio. Di conseguenza permettono un range di valori limitato.

Ad esempio, possono essere utilizzati per limitare l'occupazione della memoria quando la quantità di dati è molto grande o per limitare esplicitamente i valori che le variabili possono assumere.

Il tipo di dato **long** permette valori integrali molto grandi.

Per valori ancora più grandi è preferibile utilizzare il tipo di dato **`java.math.BigInteger`**.

## Il tipo char

Serve per rappresentare, ad esempio, i **singoli caratteri** dell'alfabeto.

Il tipo di dati **char** permette l'utilizzo di **caratteri Unicode**.

**Unicode** è uno standard dell'industria informatica per la rappresentazione consistente del testo. Unicode è composto da un catalogo di più di 110'000 caratteri, regole e metodologie per l'encoding e l'utilizzo dei caratteri.

Lo Unicode ha parzialmente sostituito lo standard ASCII. Ad esempio è stato implementato in tecnologie come l'XML, il Java e il Microsoft.NET.

Una lista dei caratteri Unicode è disponibile all'indirizzo:

[http://en.wikipedia.org/wiki/List\\_of\\_Unicode\\_characters](http://en.wikipedia.org/wiki/List_of_Unicode_characters)

## Tipo di dato a virgola mobile

L'**IEEE 754** è uno standard tecnico per la ***floating-point arithmetic*** che è stato definito nel 1985 dall'Institute of Electrical and Electronics Engineers (IEEE).

Lo standard definisce:

- i formati aritmetici e gli encodings per lo scambio dati;
- le regole per gli arrotondamenti;
- le operazioni disponibili;
- le condizioni d'eccezione (ad esempio DivisionByZero e Overflow).

## Tipo di dato a virgola mobile

Ogni numero è descritto da 3 interi:

- sign: **segno** (1 bit)
- e: **esponente** (8 o 11 bits)
- b: **mantissa** (23 o 52 bits)

IEEE-754, 32 bits:  $valore = (-1)^{sign} * \left(1 + \sum_{i=1}^{23} b_{23-i} * 2^{-i}\right) * 2^{(e-127)}$

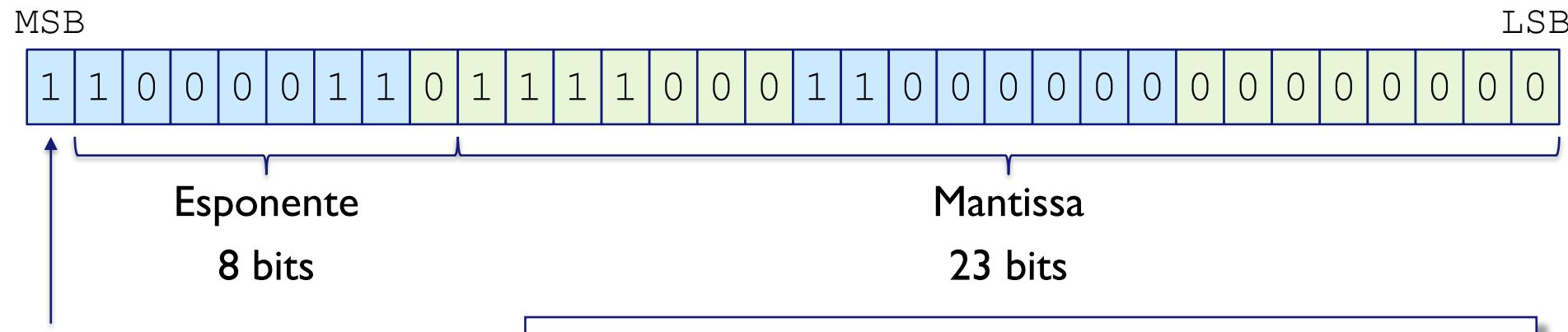
IEEE-754, 64 bits:  $valore = (-1)^{sign} * \left(1 + \sum_{i=1}^{52} b_{52-i} * 2^{-i}\right) * 2^{(e-1023)}$

Caratteristiche (valori teorici IEEE-754):

<i>Tipo di dato</i>	<i>Cifre</i>	<i>e min</i>	<i>e max</i>
Decimal 32	7	-95	+96
Decimal 64	16	-383	+384

## Tipo di dato a virgola mobile

Formato floating point IEEE-754, 32 bits (single precision):



Esempio:

Esadecimale: C3 78 C0 00

Sign bit: 1

Esponente: 10000110 = 134

Mantissa: 11110001100000000000000

$$= 2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-8} + 2^{-9} =$$

$$= 483/512 = 0.943359375$$

Valore:  $(-1)^1 \cdot (1 + 483/512) \cdot 2^{134-127} = -248.75$



## Esempi

**Binario:** 0 01111111 0000000000000000000000000000

**Sign bit:** 0      **Esponente:** 01111111 = 127

**Mantissa:** 0000000000000000000000000000 = 0

**Valore:**  $(-1)^0 * (1 + 0) * 2^{127-127} = 1.0$

**Binario:** 0 10000000 0000000000000000000000000000

**Sign bit:** 0      **Esponente:** 10000000 = 128

**Mantissa:** 0000000000000000000000000000 = 0

**Valore:**  $(-1)^0 * (1 + 0) * 2^{128-127} = 2.0$

**Binario:** 0 10000001 0000000000000000000000000000

**Sign bit:** 0      **Esponente:** 10000001 = 129

**Mantissa:** 0000000000000000000000000000 = 0

**Valore:**  $(-1)^0 * (1 + 0) * 2^{129-127} = 4.0$

**Binario:** 0 10000010 0000000000000000000000000000

**Sign bit:** 0      **Esponente:** 10000010 = 130

**Mantissa:** 0000000000000000000000000000 = 0

**Valore:**  $(-1)^0 * (1 + 0) * 2^{130-127} = 8.0$



## Esempi

**Binario:** 0 01111111 00000000000000000000000000000001

**Sign bit:** 0      **Esponente:** 01111111 = 127      **Mantissa:** 00000000000000000000000000000001 =  $2^{-23}$

**Valore:**  $(-1)^0 \cdot (1 + 2^{-23}) \cdot 2^{127-127} = 1.0000001$

**Binario:** 0 01111110 000000000000000000000000000000010

**Sign bit:** 0      **Esponente:** 01111110 = 126      **Mantissa:** 000000000000000000000000000000010 =  $2^{-22}$

**Valore:**  $(-1)^0 \cdot (1 + 2^{-22}) \cdot 2^{126-127} = 0.5000001$

**Binario:** 1 10000001 01100000000000000000000000000000

**Sign bit:** 1      **Esponente:** 10000001 = 129      **Mantissa:** 01100000000000000000000000000000 = 3/8

**Valore:**  $(-1)^1 \cdot (1 + 3/8) \cdot 2^{129-127} = -5.5$

**Binario:** 1 01111110 00011001100110011001101

**Sign bit:** 1      **Esponente:** 01111110 = 126      **Mantissa:** 00011001100110011001101 = 0.1

**Valore:**  $(-1)^1 \cdot (1 + 0.1) \cdot 2^{126-127} = -0.55$

**Binario:** 1 00000011 00101011011100101101000

**Sign bit:** 1      **Esponente:** 00000011 = 3      **Mantissa:** 00101011011100101101000 = 0.1697206

**Valore:**  $(-1)^1 \cdot (1 + 0.1697206) \cdot 2^{3-127} = -5.5 \times 10^{-38}$

## Valori a virgola mobile e precisione

Con i dati a virgola mobile **bisogna fare attenzione alla precisione.**

Ad esempio, se si esegue la somma fra due numeri float, può capitare che:

$$14573245.0f + 0.5698743f = 1.4573246e7f$$

## Tipi numerici a virgola mobile: float e double

Rappresentano un'approssimazione di  $\mathbb{R}$  (insieme dei numeri reali).

In Java:

<i>Tipo di dato</i>	<i>Cifre significative</i>	<i>Valore minimo</i>	<i>Valore massimo</i>
float	7	$10^{-45}$	$10^{38}$
double	15	$10^{-324}$	$10^{308}$

Il tipo di dato float può essere utilizzato quando serve una precisione minore o quando il range di valori deve essere limitato.

Per valori molto precisi è preferibile usare il tipo di dato ***java.math.BigDecimal***.



# Operatori relazionali

Per **confrontare due valori** e stabilire un ordinamento si usano gli operatori relazionali.

Operatore	Significato
>	maggiore
$\geq$	maggiore o uguale
<	minore
$\leq$	minore o uguale
$=$	uguale
$\neq$	diverso

Il **risultato** dell'applicazione di un operatore relazionale è un **valore booleano** true o false.

## Operatori relazionali

Gli operatori relazionali possono essere usati per confrontare valori di un **qualsiasi tipo numerico**.

Possono essere utilizzati anche per valori di **tipo char**, ma si intende un confronto fra **valori Unicode**.

Gli operatori **==** e **!=** possono essere usati anche per confrontare **valori di tipo boolean**.

Gli operatori di confronto (**<**, **<=**, **>=**, **>**) **NON possono essere utilizzati per paragonare variabili di tipo String**. Nel caso delle stringhe, i confronti **==** e **!=** vanno normalmente sostituiti con il comando **equals()**. Maggiori dettagli nel corso delle prossime lezioni.



# Esempio

```
public class EsempioOperatoriRelazionali {  
    public static void main(String[] args) {  
        int a = 10;  
        int b = 20;  
  
        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));  
    }  
}
```

# Operatori d'assegnamento

In Java, l'operatore di **assegnamento** è **=**.

Ci sono diverse **varianti** dell'operatore di assegnamento:

<i>Operatore</i>	<i>Utilizzo</i>	<i>Equivale a</i>
=	x = y	assegnazione
+=	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



# Operatori logici

Con gli operatori logici è possibile eseguire operazioni su **valori di tipo boolean**.

Operatore	Significato
& &	and
	or
!	not
^	xor



## Esempio

```
public class EsempioOperatoriLogici {  
    public static void main(String[] args) {  
        boolean b;  
  
        b = 3 > 2 || 5 < 7; // b è true  
        b = 2 > 3 || 5 < 7; // b è ancora true  
        b = 2 > 3 || 5 > 7; // ora b è false  
        b = !(3 > 2); // b è false  
        b = !(2 > 3); // b è true  
    }  
}
```

## Operatori cortocircuitati

Gli operatori `&&` e `||` di Java sono **cortocircuitati**. Significa che la seconda espressione viene valutata solo se necessario.

Esempio:

```
(x != 0) && (y / x > 1)
```

se `x = 0` allora la condizione `x != 0` restituisce `false` ed è inutile, oltre che dannoso, verificare se `y / x` sia `> 1`.

## Esempio

```
if (x == 2) {  
    if (y != 2) {  
        System.out.println("Entrambe le condizioni sono vere.");  
    }  
}
```

```
if (x == 2 && y != 2) {  
    System.out.println("Entrambe le condizioni sono vere.");  
}
```



# Operatori aritmetici

<i>Operatore</i>	<i>Significato</i>
+	addizione
-	sottrazione
*	moltiplicazione
/	divisione
%	resto della divisione (modulo)
++	pre o post incremento
--	pre o post decremento



## Operatori aritmetici

Gli **operatori aritmetici fondamentali** sono **+, -, \*, /.**

**Si applicano a tutti i tipi numerici:** byte, short, int, long, float, double.

**Si possono applicare anche ai char,** ma si applicano al loro valore **Unicode**.  
Prima dell'operazione, il carattere viene convertito in un intero corrispondente  
al valore Unicode.

Un operatore utile è l'operatore **meno unario (-)** che cambia il segno del  
valore che lo segue.

## Operatori ++ e --

L'operazione di incremento `x = x + 1;` può essere scritta in maniera più compatta con `x++;`.

Analogamente, l'espressione `x = x - 1;` equivale a `x--;`.

Questi operatori possono essere prefissi o postfissi:

- **operatore postfisso `x++`:** prima valuto `x` e poi incremento.
- **operatore prefisso `++x`:** prima incremento `x` e poi valuto.

Esempio:

```
int x = 1;

System.out.println("Post: x = " + x++); // Post: x = 1
System.out.println("Pre: x = " + ++x); // Pre: x = 3
```

## Esponenziazione e resto della divisione

^ e \*\* non sono operatori d'esponenziazione. In sostituzione è possibile utilizzare la funzione **Math.pow()**, che discuteremo in seguito.

% è l'operatore modulo che restituisce il resto della divisione intera tra due numeri. Se  $r = x \% y$ , allora esiste un  $q$  tale che  $x = q * y + r$ .

In Java, l'operatore modulo funziona anche per valori a virgola mobile:

```
int x1 = 5 / 2;
System.out.println(x1); // 2
int x2 = 5 % 2;
System.out.println(x2); // 1
double x3 = 5.0 / 2.0;
System.out.println(x3); // 2.5
double x4 = 5.0 % 2.0;
System.out.println(x4); // 1.0
```

## Esempi di operatori su tipi integrali

```
int i = 2, j = 3, k = 4;

i++; // i = i + 1
--j; // j = j - 1
k *= i + j; // k = k * (i + j)
System.out.println("Step 1: i=" + i + ", j=" + j + ", k=" + k);

i = k % j; // i = k modulo j
j = k / 3; // integer division
System.out.println("Step 2: i=" + i + ", j=" + j + ", k=" + k);

i = 2;
j = i++; // j = i; i = i + 1;
k = ++i; // i = i + 1; k = i;
System.out.println("Step 3: i=" + i + ", j=" + j + ", k=" + k);
```

### Output:

```
Step 1: i=3, j=2, k=20
Step 2: i=0, j=6, k=20
Step 3: i=4, j=2, k=4
```



## Esempi di operatori su tipi integrali

```
int a = 15;
int b = 24;

int c1 = b - a + 7;
int c2 = b - a - 4;
int c3 = b % a / 2;
int c4 = b * a / 2;
int c5 = b / (2 * a);

System.out.println("c1 = " + c1);
System.out.println("c2 = " + c2);
System.out.println("c3 = " + c3);
System.out.println("c4 = " + c4);
System.out.println("c5 = " + c5);
```

### Output:

```
c1 = 16
c2 = 5
c3 = 4
c4 = 180
c5 = 0
```



## Esempi di operatori su tipi a virgola mobile

```
double a = 15;  
double b = 24;  
  
double c1 = b - a + 7;  
double c2 = b - a - 4;  
double c3 = b % a / 2;  
double c4 = b * a / 2;  
double c5 = b / (2 * a);  
  
System.out.println("c1 = " + c1);  
System.out.println("c2 = " + c2);  
System.out.println("c3 = " + c3);  
System.out.println("c4 = " + c4);  
System.out.println("c5 = " + c5);
```

### Output:

```
c1 = 16.0  
c2 = 5.0  
c3 = 4.5  
c4 = 180.0  
c5 = 0.8
```



# Precedenze degli operatori

Priorità	Operatori	Operazione	Associatività
1	[ ] ( ) . . ++ --	Accesso elemento di un array Invocazione metodo Accesso ad un membro di un oggetto Post-incremento e post-decremento	Sinistra
2	++ -- + - ~ !	Pre-incremento e pre-decremento + / - unario Bitwise NOT Boolean NOT (logico)	Destra
3	(tipo) new	Type cast Creazione oggetto	Destra
4	* / %	Moltiplicazione, divisione, resto	Sinistra
5	+ - +	Somma, sottrazione Concatenazione di stringhe	Sinistra
6	<< >> >>>	Bit shift a sinistra (signed) Bit shift a destra (signed) Bit shift a destra (unsigned)	Sinistra



# Precedenze degli operatori

Priorità	Operatori	Operazione	Associatività
7	< <= > >= instanceof	Minore di, minore o uguale a Maggiore di, maggiore o uguale a Controllo tipo referenza	Sinistra
8	== !=	Uguale a Diverso da	Sinistra
9	&	Bitwise AND, boolean AND (logico)	Sinistra
10	^	Bitwise XOR, boolean XOR (logico)	Sinistra
11		Bitwise OR, boolean OR (logico)	Sinistra
12	&&	Boolean AND (logico)	Sinistra
13		Boolean OR (logico)	Sinistra
14	? :	Operatore ternario	Destra
15	= *= /= += -= %= <<= >>= >>>= &= ^=  =	Assegnazione Assegnamento combinato (operazione ed assegnamento)	Destra

## Regole di precedenza

Se **operatori con la stessa precedenza** sono **nella stessa espressione** e non ci sono parentesi:

- operatori **unari** e **operatori di assegnazione** sono valutati da **destra a sinistra**,
- tutti gli **altri operatori** sono valutati **da sinistra a destra**.

Esempi:

- $A * B / C$  è come scrivere  $(A * B) / C$
- $A = B = C$  è come scrivere  $A = (B = C)$
- $A * B + C$  è diverso da  $A * (B + C)$

## Letterali

“Un **letterale** è la rappresentazione in codice sorgente di un valore di tipo primitivo, una stringa o il valore null.”

— fonte: *Java language specification*

Esempi:

0

34528

22.5

0x00FF00FF

5e-28

true

'm'

"Hello Students"

null

## Letterali

Un **letterale numerico** è di tipo **double** se contiene la **virgola ‘.’**. Altrimenti è di tipo **int**.

```
double x1 = 5.3456;
double x2 = 5.0;
double x3 = 5.;

int y4 = 5;
int y5 = 2;

double z1 = 5. / 2.;
double z2 = 5 / 2.;
double z3 = 5. / 2;
double z4 = 5 / 2;
int z5 = 5 / 2;

System.out.println(z1); // 2.5
System.out.println(z2); // 2.5
System.out.println(z3); // 2.5
System.out.println(z4); // 2.0
System.out.println(z5); // 2
```

## Letterali

Un letterale intero è di tipo **long** se termina con la **lettera** ‘**I**’ o ‘**L**’. Altrimenti è di tipo **int**.

Un letterale floating point è di tipo **float** se termina con la **lettera** ‘**f**’ o ‘**F**’. Altrimenti è di tipo **double**. Opzionalmente può terminare con la lettera ‘**d**’ o ‘**D**’.

Il prefisso ‘**0x**’ indica valori **esadecimale** e ‘**0b**’ indica valori **binari** (supportati a partire da Java 7).

Per i tipi floating point si può utilizzare la **notazione scientifica** utilizzando la **lettera** ‘**e**’ o ‘**E**’.

## Letterali: caratteri e stringhe

I letterali di tipo **char** vanno sempre messi tra **apici** (esempio: 'm').

I letterali di tipo **String** vanno sempre messi tra **virgolette** (esempio: "Ciao").

Per i caratteri speciali, Java supporta le **escape sequences**:

\b	backspace
\t	tab
\n	line feed
\f	form feed
\r	carriage return
\"	double quote
\'	single quote
\\"	backslash
\uXXXX	unicode number

## Letterali, esempi

```
boolean result = true;
char capitalC = 'C';
char unicodeC = '\u0043';
byte b = 100;
short s = 10000;
int i = 100000;
long l = 100000000000L;
```

```
int decVal = 26;
int hexVal = 0x1A;
int binVal = 0b11010;
double d1 = 123.4;
double d2 = 1.234e2;
float f1 = 123.4f;
String str = "Buongiorno\nna\ttutti";
```

## Letterali, esempi

### L'istruzione

```
float f = 12.3;
```

dà un errore di compilazione perché 12.3 è un letterale di tipo double. Per correggere l'errore bisogna scrivere:

```
float f = 12.3f;
```

### L'istruzione

```
byte b = 2566;
```

dà un errore di compilazione perché il valore massimo ammesso per il tipo byte è 127.

## Valori di default

Tipo	Valore default
boolean	false
byte	0
short	0
int	0
long	0L

Tipo	Valore default
float	0.0f
double	0.0d
char	'\u0000'
String	null

Nel caso delle **variabili locali a un blocco di codice, il compilatore non assegna automaticamente un valore di default**. Prima di utilizzare la variabile è quindi importante assicurarsi che venga assegnato un valore. Il compilatore di Java segnala questo tipo di problema con un errore.

## Conversioni fra differenti tipi di dato

È possibile **convertire valori da un tipo di dato ad un altro**. In inglese questa conversione viene chiamata **type casting**.

Alcune conversioni sono **implicite**, altre devono essere specificate in **maniera esplicita**.

Le conversioni di tipo vengono definite “**widening**” (ad esempio da int a long) o “**narrowing**” (ad esempio da int a short).

Tutte le conversioni “**narrowing**” necessitano di **type cast esplicito**, tranne nel caso di determinate assegnazioni di letterali.

```
byte b = 100;  
short s = 555;
```

## Conversioni fra differenti tipi primitivi

Fra le conversioni “widening” (che non necessitano di cast), ci sono conversioni che possono causare **perdita di precisione**, come ad esempio da int a float.

```
int ii = 33554435;  
float ff = ii;  
  
System.out.println("Int: " + ii);  
System.out.println("Float: " + ff);
```

Output:

```
Int: 33554435  
Float: 3.3554436E7
```

Le conversioni dai tipi a virgola mobile ai tipi interi eliminano l'informazione dopo la virgola; viene eseguito un **troncamento (arrotondamento verso il basso)**.

```
double dd = 1234.56;  
short ss = (short) dd;  
  
System.out.println("Double: " + dd);  
System.out.println("Short: " + ss);
```

Output:

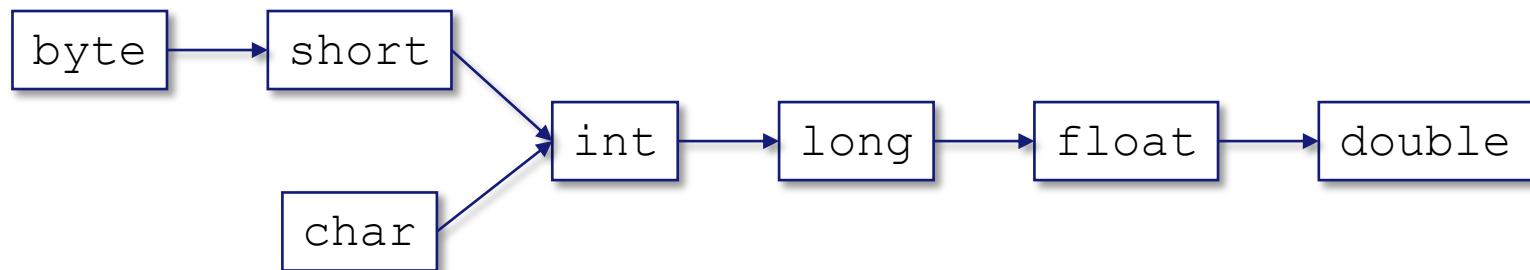
```
Double: 1234.56  
Short: 1234
```

## Conversioni implicite

Le **conversioni implicite** vengono **effettuate automaticamente dal compilatore**.

Qualunque valore numerico è assegnabile a qualunque variabile numerica il cui tipo abbia un **range di valori più ampio** o, nel caso di valori a virgola mobile, una **precisione maggiore**.

In generale, le conversioni implicite consentite ed effettuate dal compilatore in modo automatico sono:



Viceversa, le conversioni devono essere eseguite in maniera esplicita poiché causano una perdita di informazioni.

## Conversioni implicite: esempi

Un `char` può essere utilizzato come un `int`:

```
char car1 = 'a'; // La lettera a è il numero 97
char car2 = 'b'; // La lettera b è il numero 98
int var = car1 + car2; // È come se stesse facendo 97 + 98
System.out.println(var); // Stamperà 195
```

Un `float` può essere memorizzato in una variabile di tipo `double`:

```
float a = 4.5f;
double b = a;
System.out.println(b);
```

Altri esempi:

```
int var1 = 10;
long var2 = var1;
System.out.println(var2);
```

```
short var1 = 120;
int var2 = var1;
System.out.println(var2);
```

## Conversioni implicite nelle espressioni

In una espressione possono essere coinvolte più variabili di tipi diversi. In questo caso **le variabili sono automaticamente promosse ad una forma più estesa** per non causare perdita di informazioni.

```
int a = 4;
float b = -1.0f;
double c = 2.4;
int d = 9;
double espr = (a - b) * c / ((d * 4) + 6.7f);
```

Le variabili `a`, `b` e `d` ed i letterali `4` e `6.7f` vengono convertiti implicitamente al tipo più esteso, in questo caso `double`.

## Conversioni esplicite

Il casting è una **conversione esplicita utilizzata per forzare la conversione di un tipo di dato ad un altro**. Questa conversione può però causare una **perdita di informazioni**.

```
int a = 32765;  
short b = (short) a;
```

Java ha un **comportamento preventivo** vietando qualunque operazione possa far nascere dei dubbi. Le conversioni esplicite sono eseguite solamente quando non c'è perdita di informazione.



## Conversioni esplicite: esempi

```
int a;
double x;
short b;
a = 17;
x = a; // OK: a è convertito in un double
b = a; // Errore: cast implicito non possibile
b = (short) a;
short c = 20;
```

```
int k = 47;
float x, y, z, zz;
double u, v, w;
x = k / 3; // Divisione tra interi seguita da conversione a float
y = (float) k / 3; // Divisione tra float
z = (float) k / (float) 3; // Divisione tra float
zz = (float) (k / 3); // Divisione tra interi seguita da cast a float
u = k / 3.0; // Divisione tra double
v = k * 1. / 3; // Divisione tra double
w = -2.0e4; // Notazione scientifica
System.out.println("x=" + x + ", y=" + y + ", z=" + z + ", zz=" + zz);
System.out.println("u=" + u + ", v=" + v + ", w=" + w);
```

## Conversioni esplicite: esempi

```
int var1 = 32768;  
short var2 = (short) var1;  
System.out.println(var2);
```

Il casting provocherà una perdita di informazione; il tipo `short` ha un range di valori più piccolo rispetto al tipo `int` e verrà eseguito un ***wrap around***. La variabile `var2` assumerà quindi il valore -32768.

# Riepilogo

- Variabili e costanti
- Tipi di dato primitivi
- Tipi numerici interi
- Tipi numerici a virgola mobile
- Operatori relazionali
- Operatori d'assegnamento
- Operatori logici
- Operatori aritmetici
- Pre e post incremento
- Resto della divisione ed esponenziazione
- Precedenza degli operatori
- Letterali
- Valori di default
- Conversioni implicite ed esplicite