

- └ Tipi di dato primitivi
 - └ Assegnare valori alle variabili

Letterali

- ▶ A differenza di una variabile, il **valore di un letterale non cambia**
 - ▶ Letterali di tipo numero come 5.0 (costante reale) e 5 (costante di tipo intero)
 - ▶ Letterali di tipo carattere come 'A', 'B' e '*'
 - ▶ Letterali di tipo Booleano, ovvero le costanti `true` e `false`
 - ▶ Letterali di tipo stringa: come "Contatore: "

- └ Tipi di dato primitivi
- └ Assegnare valori alle variabili

Scambio di variabili

- ▶ Operazione molto frequente
- ▶ Non può essere ridotta a esecuzione di

```
x = y;
```

```
y = x;
```

- ▶ Si perde valore iniziale di `x` ed entrambe le variabili hanno al termine dell'esecuzione il valore di `y` prima dello scambio
- ▶ Può essere eseguita da istruzioni (usando una variabile di appoggio):

```
z = x;
```

```
x = y;
```

```
y = z;
```

- └ Tipi di dato primitivi
 - └ Assegnare valori alle variabili—Assegnare valori iniziali alle variabili

Assegnare valori iniziali alle variabili

- ▶ Opzionale ma consigliato

- ▶ Sintassi

tipo var-1 = esp-1, var-2 = esp-2, ...;

- ▶ Esempio

```
int punteggio = 0;  
char lettera = 'p';  
double altezza = 12.34, base = 5.1;
```

- ▶ Ottenuti combinando operatore di assegnazione con operatore aritmetico

```
int punteggio = 0;  
System.out.println( "Punteggio: "+punteggio );  
punteggio += 5;  
System.out.println( "Punteggio: "+punteggio );
```

- ▶ Istruzione

```
punteggio += 5;  
equivalente a  
punteggio = punteggio+5;
```

- ▶ Espressione a destra trattata come singola unità

- ▶ `x *= a+b;`
equivalente a
`x = x*(a+b);`
e non a
`x = x*a+b;`

- ▶ Due tipi: quelli su una singola riga e quelli su righe multiple
- ▶ Sintassi

// commento limitato a una singola linea


/ commento distribuito su piu' linee
senza limiti sul numero di righe */*

- ▶ Esempio

```
boolean primo; // indica se il numero e' primo
```

```
/* La variabile primo indica se il numero e'  
primo: la primalita' viene determinata dividendo  
il numero per tutti i suoi possibili divisori. */  
boolean primo;
```

- ▶ **Conversione**: necessaria per assegnare valore di un tipo a variabile di tipo diverso
 - ▶ Cambia il tipo del valore, non della variabile
- ▶ **Implicita** (ovvero automatica): si assegna valore di tipo “più basso” a variabile di tipo “più alto” nella gerarchia

- 
1. double
 2. float
 3. long
 4. int
 5. short
 6. byte

| NOME | TIPO | MEMORIA |
|---------|------------|---------|
| byte | intero | 1 byte |
| short | intero | 2 byte |
| int | intero | 4 byte |
| long | intero | 8 byte |
| float | reale | 4 byte |
| double | reale | 8 byte |
| char | carattere | 2 byte |
| boolean | vero/falso | 1 byte |

- ▶ Esempio

```
double x;  
int n = 5;  
x = n; // x contiene 5.0, n non viene modificata  
System.out.println( "x: " + x );
```

- └ Tipi di dato primitivi
 - └ Conversione di tipo—Conversione implicita

Valori rappresentabili:

- *byte* – interi nell'intervallo [-128,127] (8 bit)
- *short* – interi nell'intervallo [-32768, 32767] (16 bit)
- *int* – interi nell'intervallo [-2147483648, 2147483647] (32 bit)
- *long* – interi nell'intervallo [-2^{63} , $2^{63}-1$] (64 bit)
- *float* – razionali tra 10^{-45} e 10^{+38} (approx.) in valore assoluto (32 bit)
- *double* – razionali tra 10^{-324} e 10^{+308} (approx.) in valore assoluto (64 bit)
- *char* – caratteri dell'alfabeto Unicode 2.0 (16 bit)
- *boolean* – un valore tra i due seguenti {*true*, *false*} (8 bit)

Tipo di un'espressione

| Tipo del Risultato | Tipo degli operandi |
|--------------------|--|
| long | Nessun operando è un float o un double (aritmetica intera); ma almeno uno è un long. |
| int | Nessun operando è un float o un double (aritmetica intera); nessun operando è un long. |
| double | Almeno un operando è un double. |
| float | Almeno un operando è un float; nessun operando è un double. |

- ▶ Operandi tutti dello **stesso tipo**: tipo del valore di ritorno è **quello degli operandi**
- ▶ Operandi di **tipo diverso**: il tipo del valore di ritorno è quello **più alto nella gerarchia**
- ▶ Esempio

```
double a;  
int n = 2;  
double x = 5.1;  
double y = 1.33;  
a = (n*x)/y;  
System.out.println( "a: "+a );
```



Tipo di un'espressione (esempio 1)

```
double a;  
int n = 2;  
double x = 5.1;  
double y = 1.33;  
a = (n*x)/y; // <- cosa avviene  
System.out.println( "a: "+a );
```

- ▶ il valore di `n` viene convertito in `double`: 2.0
- ▶ moltiplicato per `x` (ovvero 5.1)
- ▶ diviso per `y` (ovvero 1.33)
- ▶ il risultato viene assegnato ad `a`

Tipo di un'espressione (esempio 2)

```
double a;  
int n = 2;  
int x = 5; // <- x di tipo int, non double  
double y = 1.33;  
a = (n*x)/y; // <- cosa avviene  
System.out.println( "a: "+a );
```

- ▶ il valore di `n` viene moltiplicato per `x` (ovvero 5)
- ▶ il risultato è 10, non 10.0
- ▶ il risultato viene convertito in `double` (ovvero 10.0) e diviso per `y` (ovvero 1.33)
- ▶ il risultato viene assegnato ad `a`

Conversione implicita e perdita di precisione

- Conversioni da tipi interi a tipi in virgola mobile: **perdita di precisione**

- Non tutti i valori di tipo `int` sono rappresentati nel tipo `float`

```
int x = 2109876543;
float y = x;
int z = (int)y;
System.out.println( "x: "+x );
System.out.println( "y: "+y );
System.out.println( "z: "+z );
```

```
x: 2109876543
y: 2.10987648E9
z: 2109876480
```

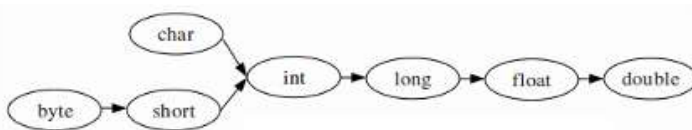
- └ Tipi di dato primitivi
 - └ Conversione di tipo—Conversione esplicita

Conversione esplicita

- ▶ Necessaria per assegnare valore di tipo “più alto” a variabile di tipo “più basso”
 - ▶ `intero = reale;`
genera errore
possible loss of precision
found : double
required: int
- ▶ Sintassi `var 1 = (tipo)var 2;`
- ▶ Esempio

```
numeroIntero = (int)numeroReale;  
carattere = (char)numeroIntero;
```

Conversione **implicita** --> se si segue il senso delle frecce
Conversione **esplicita** --> altrimenti



NOTA: L'intervallo di valori che può essere rappresentato da un **float** o **double** è molto più grande dell'intervallo che può essere rappresentato da un **long**. Sebbene si possa perdere cifre significative quando si converte da long a float, si tratta comunque di un'operazione di "allargamento" perché l'intervallo è più ampio, quindi si tratta di conversione implicita.

Troncamento

- ▶ Quando si converte un valore di tipo in **virgola mobile** in un tipo **intero**, la parte decimale viene ignorata

```
int numeroEuro;  
double conto = 26.99;  
numeroEuro = (int)conto;  
System.out.println( "Euro: "+numeroEuro );  
// stampa 26, non 27
```

- ▶ Non tutti i **numeri reali** sono rappresentati in **modo esatto**: troncamento può causare perdita di precisione

```
double f = 4.35;  
int n = (int)(100*f);  
System.out.println( "n: "+n ); // 434, non 435
```

Divisione reale e divisione intera

- ▶ Se almeno uno dei due operandi è di tipo `float` o `double`, risultato quello aspettato
- ▶ Se entrambi operandi sono di tipo intero, parte frazionaria ignorata

```
System.out.println( "5/4.0: "+5/4.0 );  
// OK: stampa 1.25  
System.out.println("5/4: "+5/4);  
// stampa 1, troncando parte frazionaria
```

Caratteri come interi

- ▶ In ASCII e Unicode codice cifre diverso da loro valore numerico

```
char cifra = '6';  
int cifraIntera = cifra; // 54, non 6  
System.out.println( "Cifra: "+cifraIntera );
```

- ▶ Però codici cifre sono numeri interi consecutivi a partire da 48

```
char cifra = '6';  
int cifraIntera = cifra-48; // 6  
System.out.println( "Cifra: "+cifraIntera );
```


Operatore modulo

- ▶ %: restituisce **resto** divisione primo operando per secondo
 - ▶ Il valore di ritorno di $20\%6$ è 2
- ▶ Diverse applicazioni
 - ▶ Consente di contare modulo un certo valore n
 - ▶ Ad esempio, 0, 1, 2, 0, 1, 2, ...
 - ▶ Consente di decidere se un numero multiplo di un altro
 - ▶ Primo algoritmo per massimo comun divisore
 - ▶ Parte integrante di algoritmo
 - ▶ Algoritmo di Euclide

Precedenze e parentesi

- ▶ Espressioni seguono normali regole di precedenza
 1. Operatori **unari** (ovvero con un solo argomento) +, -, ++, --.
 2. Operatori **binari** (ovvero con due argomenti) *, /, %.
 3. Operatori **binari** +, -.
- ▶ Parentesi forzano precedenza

```
int x = 10, y = 2;  
double f = 0.2;  
double risultato = (x+y)*f;  
System.out.println( "(x+y)*f: "+risultato );  
risultato = x+(y*f);  
System.out.println( "x+(y*f): "+risultato );
```

Note sulle espressioni

- ▶ Se la precedenza è ovvia le parentesi non importano. . .
- ▶ . . . ma si possono mettere se rendono l'espressione più chiara da leggere
- ▶ gli spazi fra gli operatori, le parentesi e le espressioni non contano.

- └ Tipi di dato primitivi
- └ Operatori di incremento e decremento

Operatori di incremento e decremento

- ▶ Aumentano o diminuiscono di uno il valore di variabile intera
- ▶ `punteggio++`;
equivale a
`punteggio = punteggio+1`;
e `punteggio--`;
equivale a
`punteggio = punteggio-1`;
- ▶ Operatore **precede** (**segue**) operando: valore variabile modificato **prima** (**dopo**) di essere usato

```
int contatore = 5;  
int n = 2*(++contatore);  
int m = 2*(contatore++);
```



Operatori di incremento e decremento

- ▶ Aumentano o diminuiscono di uno il valore di variabile intera

- ▶ `punteggio++`;

equivale a

```
punteggio = punteggio+1;
```

e `punteggio--`;

equivale a

```
punteggio = punteggio-1;
```

- ▶ Operatore **precede** (**segue**) operando: valore variabile modificato **prima** (**dopo**) di essere usato

```
int contatore = 5;  
int n = 2*(++contatore);  
int m = 2*(contatore++);  
// n vale 12  
// m vale 12  
// contatore vale 7
```