Tipi di dati primitivi

Programmazione
Corso di laurea in Comunicazione digitale

La nozione di tipo di dato

- Il tipo del dato consente di esprimere la natura del dato
- Indica il modo con cui verrà interpretata la sequenza di bit che rappresenta il dato
  - La stessa sequenza può rappresentare un intero o un carattere ad esempio
- Determina il campo dei valori che un dato può assumere
- Specifica le operazioni possibili sui dati

Tipi di dati e Java

- Java è un linguaggio ***fortemente tipizzato***
- Il tipo di ogni variabile o espressione può essere identificato leggendo il programma ed è già noto al momento della compilazione
  - È obbligatorio dichiarare il tipo di una variabile prima di utilizzarla
  - Durante la compilazione sono effettuati tutti i controlli relativi alla compatibilità dei tipi e sono stabilite eventuali conversioni implicite.
- Dopo la dichiarazione non è possibile assegnare alla variabile valori di tipo diverso
  - Salvo i casi di ***shadowing*** della ***visibilità*** in cui lo stesso identificatore può essere usato per indicare variabili di tipo diverso

Dichiarazione e inizializzazione

- Dichiarazione di variabili

```
int x, y;
String nome;
MiaClasse nuova_variabile;
```

- Inizializzazione di variabili mediante *l'operatore di assegnamento* (=)

```
int x = 5;
String nome = new String("pallone");
nuova_variabile = new MiaClasse();
```


Tipi di dato

- Ogni tipo di dato è individuato da un ***identificatore*** che è una parola riservata per i tipi semplici
  - `int`, `double`, `char`, `MiaClasse`
- Definisce un ***insieme di valori*** letterali possibili
  - Ad esempio: `3`, `3.1`, `'c'`, `un indirizzo valido`
- E un insieme di ***operazioni lecite***
  - Ad esempio: `+` `*` `mioMetodo()`

In Java i dati sono di due categorie di tipi:

- tipi primitivi, detti anche *semplici*
- tipi di oggetti o *riferimenti a oggetti*

Tipi di dati primitivi o semplici

8 tipi di dati ***primitivi*** per rappresentare:

- numeri interi
  - `byte`, `short`, `int`, `long`
- numeri decimali in virgola mobile
  - `float`, `double`
- i caratteri
  - `char`
- i valori booleani
  - `boolean`

Tipi di dati primitivi numerici

- La differenza tra i diversi tipi per dati numerici consiste nello spazio di memoria che viene utilizzata per la rappresentazione
  - E quindi nell'intervallo di valori che possono rappresentare

Tipo	Memoria in byte	Valore min	Valore max
byte	1	-128 (-2^7)	127 (2^7-1)
short	2	-32,768 (-2^{15})	32,767 (2^{15}-1)
int	4	-2,147,483,648	2,147,483,647
long	8	-2^{63}	2^{63}-1
float	4	+/- 1.4 x 10^{-45}	+/- 3.4 x 10^{38}
double	8	+/- 4.9 x 10^{-324}	+/- 1.8 x 10^{308}

Ordini di grandezza degli interi

tipo	occ	numero di combinazioni		circa	ordine
byte	1	2^8	256		
		2^{10}		10^3	mille
short	2	2^{16}	65536		
		2^{20}		10^6	milione
		2^{30}		10^9	miliardo
int	4	2^{32}	4.294.967.296		
		2^{40}		10^{12}	mille miliardi
		2^{50}		10^{15}	milione di miliardi
		2^{60}		10^{18}	miliardo di miliardi
long	8	2^{64}	1.844.672.545.073.135.616		

Espressioni

- Una ***espressione*** è una combinazione di operatori e operandi che ha un tipo e un valore
- Esempi:
- I letterali, come: `125`, `variabile_x`
- Le variabili costituiscono un'espressione il cui valore è il contenuto della variabile stessa
- Si possono combinare espressioni semplici, come:
  - `i + j`
- Le chiamate dei metodi costituiscono un'espressione il cui valore è il risultato riportato dal metodo
  - `"aiuto".toUpperCase()` computa e riporta la stringa `"AIUTO"`

Espressioni aritmetiche

- Una ***espressione aritmetica*** calcola valori numerici mediante operatori aritmetici:

somma	+
sottrazione	-
moltiplicazione	*
divisione	/
resto	%

- Se uno o entrambi gli operandi di un operatore sono di tipo virgola mobile, il risultato è di tipo virgola mobile

Operatori sui tipi aritmetici

tipo	simbolo	operazione	esempio
`float`, `double`	+	somma	4.50e01 + 5.30e00 = 5.03e01
	-	sottrazione	6.57e02 – 5.7e01 = 6.00e02
	*	moltiplicazione	7e03 * 3.0e00 = 2.1e04
	/	divisione	9.6e01 / 2e01 = 4.8e00
`byte`, `short`, `int`, `long`	+	somma	45 + 5 = 50
	-	sottrazione	657 – 57 = 600
	*	moltiplicazione	7000 * 3 =21000
	/	divisione	10 / 3 = 3
	%	resto	10 % 3 = 1

Operatori su tipi aritmetici interi

- Qual'è il tipo del valore dell'espressione
  `i + j`
  se `i` e `j` sono entrambe di tipo:

  `int` allora `i + j` è di tipo `int`
  `long` allora `i + j` è di tipo `long`
  `short` allora `i + j` è di tipo `int`
  `byte` allora `i + j` è di tipo `int`

Divisione e resto

- Se entrambi gli operandi dell'operatore `/` sono interi, il risultato è intero e la parte decimale è persa

```
14 / 3      uguale a   4
8 / 12      uguale a   0
```

- L'operatore resto `%` riporta il resto della divisione

```
14 % 3      uguale a   2
8 % 12      uguale a   8
```

- Esempio `Divisione.java`

Conversioni implicite

- Dati
  `int y; long x, z;`
  e l'istruzione di assegnamento
  `x = y + z;`

- Il risultato dell'espressione `y + z` è di tipo `long`
- La variabile `y` viene convertita implicitamente nel corrispondente valore di tipo `long` prima di effettuare l'operazione
- In generale i valori di un tipo più ristretto possono essere promossi ad un tipo più ampio

Conversioni implicite

- Dati
  `int y, z; long x;`
  e l'istruzione di assegnamento
  `x = y + z;`

- Il risultato dell'espressione `y + z` è di tipo `int`

- Il valore di `y + z` viene promossa al tipo `long` prima di effettuare l'assegnamento

Conversioni implicite tra `int` e `long`

	x	y	z	y+z	x = y+z	promozioni
1	int	int	int	int	int	nessuna
2	long	int	int	int	long	y + z
3	long	long	int	long	long	z
4	long	int	long	long	long	y
5	long	long	long	long	long	nessuna
6	int	long	int	long	impossibile	
7	int	int	long	long	impossibile	
8	int	long	long	long	impossibile	

Conversioni esplicite

- E' possibile convertire un tipo ampio in uno piu' ristretto ma deve essere fatta su esplicita richiesta
- Forzatura o cast

```
(nome_tipo) espressione
```

```
int x;
long y;
x = (int) y
```

- La conversione di un tipo in uno più ristretto può portare a perdita di informazione
- L'utilizzazione dell'operatore di cast implica la responsabiltà del programmatore

Esempi

1.
```
int x;
long y, z;
x = (int) (y + z);
```

2.
```
int x, y;
long z;
x = (int) (y + z);
```
oppure
```
x = y + (int)z;
```

3.
```
int x, z;
long y;
x = (int) y + z;
```

Notazione scientifica e precisione

- I tipi che rappresentano numeri decimali possono essere visualizzati in diversi modi:
  - `3.14159712` `9.0` `0.5e+001` `-16.3e+002`
  - Dove `e` indica la notazione scientifica in potenze di 10 (*notazione-e*) e separa il numero dall'esponente cui elevare la base 10
  - La velocità della luce è di 299,792,458 m/sec si può scrivere come `2.9979245e8` in Java
- I tipi `float` e `double` si archiviano come valori approssimati
  - Circa le prime 7 cifre decimali possono essere archiviate per i tipi `float` e 15 per i `double`

Conversioni implicite ed esplicite

- Conversioni implicite senza perdita di informazione
  `int` ⇒ `long` ⇒ `float` ⇒ `double`
- Non tutti i valori `int` e `long` sono rappresentati da float e double. Può esserci perdita di precisione dovuta alle approssimazioni

```
int x = 2109876543;   x vale  2109876543
float y = x;          y vale  2.10987648E9
int z = (int) y;      z vale  2109876480
```


Conversioni implicite nelle espressioni

- In espressioni aritmetiche con tipi `int`, `long`, `float` e `double`

`int`	Aritmetica intera: nessun operando e' un `float/double` o `long`
`long`	Aritmetica intera: nessun operando e' un `float/double` ma almeno uno e' `long`
`float`	almeno un operando e' `float`; e nessuno e' `double`
`double`	almeno un operando e' `double`

Intervalli numerici e precisione

- I tipi interi (`byte`, `short`, `int`, `long`) rappresentano numeri interi in un dato intervallo
  - Per il tipo `int`, le costanti `Integer.MIN_VALUE` e `Integer.MAX_VALUE` danno gli estremi dell'intervallo
- I tipi in virgola mobile (`float`, `double`) rappresentano i numeri con una precisione finita e introducono approssimazioni
- Problemi di **overflow** e **perdita di precisione**
  - Es: `Overflow.java` e `Precisione.java`

I caratteri

- Una variabile `char` contiene un singolo carattere dell'insieme dei caratteri ***Unicode***
  - Un insieme di caratteri (ASCII, Unicode, …) è una tabella rappresentabile come una *lista ordinata*: a ogni carattere corrisponde uno e un solo numero
- L'insieme ***Unicode*** usa 16 bits per rappresentare un carattere, ammettendone quindi 65.536 diversi
  - E' uno standard internazionale e contiene caratteri e simboli per molti diversi linguaggi
- I caratteri letterali sono delimitati dal carattere `'`

```
'a'   'X'   '7'   '$'   ','   '\n'
```

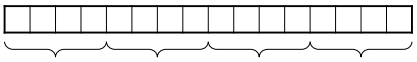

I caratteri ASCII

- L'insieme dei caratteri *ASCII* è più vecchio e più piccolo dell'Unicode, ma ancora in uso
- L'insieme dei caratteri *ASCII* è un sottoinsieme dell'insieme Unicode che comprende:

maiuscole	A, B, C, …
minuscole	a, b, c, …
punteggiatura	virgola, punto, punto e virgola, …
cifre	0, 1, 2, …
simboli speciali	&, |, \, …
caratteri di controllo	ritorno, tabulazioni, ...

I caratteri UNICODE

- Sono codificati tra i valori `\u0000` e `\uffff`
- La sequenza di escape `\u` indica che il numero che segue è un carattere ***Unicode*** e i numeri sono espressi in esadecimale
  - ***Numeri esadecimali*** sono rappresentati in base 16 quindi mediante le cifre 0-9 e a-f. 16 cifre in tutto appunto



ciascuna delle 16 cifre esadecimali rappresenta una delle 16 possibili configurazioni di 4 bit

La rappresentazione dei caratteri

Formato ASCII esteso

- Utilizza 8 bit, quindi rappresenta 256 valori
- La prima metà è il formato ASCII, ottenuta utilizzando solo 7 bit
- La seconda metà rappresenta vari caratteri speciali
  - Le lettere accentate o con segni particolari: umlaut o cediglie …
- ISO 8859 del 1980 stabilisce lo standard per il formato ASCII esteso a 8 bit
  - 0-127 ISO-Latin 1 (il vecchio ISO 646)
  - 128-159 non sono usati
  - 160-255 i caratteri speciali
- Ma 256 caratteri non sono comunque sufficienti per rappresentare i caratteri di tutte le lingue

Formato ISO10646

- ISO10646 è una collezione di 2^{32} caratteri organizzati in un ipercubo a 4 dimensioni
  - 256 gruppi di 256 piani di 256 righe di 256 caratteri di 8 bit (g,p,r,c)
  - Il formato ***Unicode***, chiamato anche *Basic Multilingual Plane*, è (0,0,r,c) e rappresenta tutti i set di caratteri inclusi il cinese, il giapponese e il coreano
- *Encoding* è la funzione che mappa un codice in una sequenza di byte per la trasmissione o l'archiviazione
  - ***Quoted Printable***: i caratteri tra 128 e 255 sono rappresentati con 3 byte di cui il 1° è il segno =, gli altri contengono il valore del codice in esadecimale
    - è diventa =E8
  - **UCS-2** trasmette solo i due piani utilizzati da UNICODE
  - **UTF8** inoltre li trasmette in opportune sequenze di byte a 8 bit

Il tipo boolean

- Una variabile di tipo `boolean` rappresenta una grandezza a due stati
  - un interruttore è ***acceso*** o ***spento***
- Un valore `boolean` rappresenta una condizione di verità o falsità
- Le parole riservate (letterali costanti) `true` e `false` sono gli unici valori ammessi per il tipo boolean

```
boolean eseguito = false;
```


Gli operatori logici

- Gli operatori che si possono applicare ai tipi `boolean` sono:
- due operatori binari che si applicano a due operandi `boolean` e riportano un valore a `boolean`
  - `&&` congiunzione o AND
  - `||` disgiunzione o OR
- un opartore unario che si applica a un operando `boolean` e riporta un valore `boolean`
  - `!` negazione o NOT

Operatore logico `not`

- L'operatore logico `!` è anche chiamato *negazione logica* o *complemento logico*
- Se una condizione booleana `a` è vera, allora `!a` è falsa; se `a` è falsa, allora `!a` è vera
- L'operatore logico **`not`** è quindi descritto dalla *tavola di verità*

a	!a
true	false
false	true

Gli operatori logici `and` e `or`

- L'espressione logica **`and`**

  `a && b`

  è vera se entrambi gli operandi `a` e `b` sono veri, ed è falsa altrimenti

- L'espressione logica **`or`**

  `a || b`

  è vera se `a` o `b` o entrambi sono veri, ed è falsa altrimenti

Tavole di verità

- Una tavola di verità mostra le possibili combinazioni di termini di valori vero/falso
- Poiché `&&` e `||` hanno due operandi ciascuno, ci sono 4 possibili combinazioni

a	b	a && b	a || b
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Espressioni booleane

- Espressioni in cui compaiono gli *operatori* di Java *di uguaglianza* o *relazionali*, che riportano valori **booleani**

`==`	**uguale**
`!=`	**non uguale**
`<`	**minore**
`>`	**maggiore**
`<=`	**minore o uguale**
`>=`	**maggiore o uguale**

- Si noti la differenza tra l'operatore di uguaglianza (`==`) e l'operatore di assegnamento (`=`)
- Vengono usate principalmente per esprimere le condizioni in istruzioni di controllo del flusso

Espressioni con operatori logici

- Gli operatori logici vengono usati per formare espressioni booleane complesse
  - Ad es condizioni in istruzioni di selezione o cicli

```
if (totale < MAX && !trovato)
   System.out.println ("Processing…");
```

- Gli operatori logici hanno relazioni di precedenza tra loro e con altri operatori

Espressioni con operatori logici

- Attenzione, da errore:

```
if ( 0 < numero < 1000) …
if ( car == 'a' || 'b') …
```

- Occorre scrivere:

```
if ( 0 < numero && numero < 1000)
if (car == 'a' || car == 'b')
```

Operatori in Java

- Operatori aritmetici
- Operatori di uguaglianza e relazionali
- Operatori logici in espressioni booleane
- Operatori di incremento e decremento
- Operatori di assegnamento
- Operatore condizionale
- La problematica della precedenza degli operatori
- L'associatività degli operatori

Precedenza di operatori

- La precedenza specifica il grado di priorità di un operatore rispetto ad un altro e quindi l'ordine secondo il quale vengono applicati.
- In che ordine vengono valutati gli operatori nell'espressione:

```
a + b * c
```

  - La moltiplicazione ha un più alto grado di precedenza rispetto alla somma, quindi ad a viene sommato il valore di `b * c`
- L'ordine di applicazione degli operatori può essere modificato mediante l'uso di parentesi (...)

```
(a + b) * c
```

  - Otterremo ora `a + b` moltiplicato per `c`

Precedenza di operatori logici

- L'operatore ! ha la massima precedenza e || la minima.

```
a && b || a && c
```

- è equivalente a

```
(a && b) || (a && c)
```

```
!a && b || a && !c
```

- è equivalente a

```
((!a) && b) || (a && (!c))
```


Tavole di verità di espressioni

- Le espressioni vengono valutate usando le tavole di verità (rispettando le precedenze: ! ha una precedenza maggiore di < , che ha precedenza su &&)

```
(totale < MAX && !trovato)
```

totale < MAX	trovato	!trovato	totale < MAX && !trovato
false	false	true	false
false	true	false	false
true	false	true	true
true	true	false	false

Legge di De Morgan

- Espressioni complesse come:

```
if (!(0<numero && numero<1000))
```

non è vero che 0 < numero e numero < 1000

possono essere semplificate per essere rese più leggibili usando la legge di De Morgan (1806-1871)

- `!(a && b)` equivale a `!a || !b`
- `!(a || b)` equivale a `!a && !b`

Semplificazione con De Morgan

- L'espressione:

```
if (!(0<numero && numero<1000))
```

si semplifica nell'espressione:

```
if (!(0<numero)||!(numero<1000))
```

e poi ancora:

```
if ((0>=numero)||(numero>=1000))
```

```
if ((numero<=0)||(numero>=1000))
```

Espressioni booleane esempi

```
boolean x, y, z;
int a;

1. (x == y) || (x == !y)
2. (x == y) && (x == !y)
3. (a >= 10) && (a <= 25)
4. (a >= 10) || (a <= 25)
5. (a < 10) && (a > 25)
6. x && (a >= 10) || (a <= 25)
7. x && ((a >= 10)&& (a <= 25))
```

Alcune condizioni sono sempre vere e altre sempre false

Altre condizioni sono vere o false secondo i valori delle variabili

Metodi predicativi

- Un *metodo predicativo* restituisce un valore di tipo `boolean`:

```
public class ContoBancario {
   public boolean e'Scoperto() {
        return this.saldo() < 0
   }
}
```

- Esempi predefiniti nella classe Character `isDigit`, `isLetter`, `isUpperCase`

Lazy evaluation

- Il risultato di un operatore è normalmente determinato dopo i relativi operandi.
- Eccezioni
- gli operatori `&&` e `||` usano lazy evaluation (valutazione *pigra* o *cortocircuitata*)
  - Se, avendo valutato solo una parte di un'espressione booleana, è già possibile determinarne il risultato, la parte rimanente non viene valutata.

```
int x, y;   …
(x > 0) && (y++ != x)
```


I flag

- Qualunque variabile che può assumere solo due valori può essere dichiarata di tipo boolean

```
private boolean coniugato;
if (coniugato) …
```

in sostituzione di

```
if (coniugato == true) …
```

- Si chiamano anche *flag*

Operatori di incremento e decremento

- Gli operatori di incremento e decremento sono operatori aritmetici unari
- L'operatore di *incremento* (`++`) aggiunge 1 al suo operando
- L'operatore di *decremento* (`--`) sottrae 1 al suo operando
- L'istruzione

```
cont++;
```

equivale all'istruzione

```
cont = cont + 1;
```


Operatori di incremento e decremento

- Operatori di incremento e decremento possono essere usati in *forma prefissa* o *postfissa* a seconda della posizione dell'operatore rispetto alla variabile
- Nella *forma prefissa* prima viene incrementata la variabile, poi viene valuta l'espressione.
- Nella *forma postfissa* prima viene valuta l'espressione, poi viene incrementata la variabile.
- Quando si usano soli in una istruzione, le due forme sono equivalenti.

`cont++;` equivale a `++cont;`

Operatori di incremento e decremento

- In un'espressione, le due forme possono avere effetti molto diversi
- Sempre la variabile viene aumentata o decrementata
- Ma il valore usato nell'espressione dipende dalla forma prefissa o postfissa:

espressione	operazione sulla variabile	valore usato nell'espressione
`cont++`	somma 1	precedente
`++cont`	somma 1	nuovo
`cont--`	sottrae 1	precedente
`--cont`	sottrae 1	nuovo

Operatori di incremento e decremento

- se `cont` contiene attualmente il valore 45, allora

```
totale = cont++;
```

assegna 45 a `totale` e 46 a `cont`

- se `cont` contiene attualmente il valore 45, allora

```
totale = ++cont;
```

assegna il valore 46 sia a `totale` sia a `cont`

Effetti collaterali

- La valutazione di espressioni contenenti `++` e `--` da luogo a effetti collaterali.

```
1. x++ + y + x
2. ++x + y + x;
3. if (x++ == --y)
       z = x + y;
   else
       z = x - y;
```


Operatori di assegnamento

- L'operando di destra di un operatore di assegnamento può essere un'espressione
- L'espressione di destra viene dapprima valutata quindi il risultato viene assegnato alla variabile, il cui precedente valore viene sovrascritto
- Nell'istruzione

```
risultato /= (totale-MIN) % num;
```

si calcola prima il valore dell'espressione

```
((totale-MIN) % num);
```

quindi si valuta `risultato / valore_espressione` e lo si assegna a `risultato`

Operatori di assegnamento

- Spesso eseguiamo operazioni di aggiornamento del valore di una variabile utilizzando sempre alcune operazioni (somma o sottrazione …)
- Alcuni operatori semplificando la notazione di *assegnamento* consentono questo processo
- Esempio:

```
num += cont;
```

equivale a

```
num = num + cont;
```


Operatori di assegnamento

operatore	esempio	equivale a
`+=`	`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`

Operatore condizionale

- L'operatore *condizionale* valuta una condizione booleana per determinare quale espressione tra due possibili vada valutata
- Il risultato dell'espressione selezionata diventa il risultato dell'operatore condizionale

```
condizione ? Espressione_1 : espressione_2
```

- Se *condizione* è vera, allora viene valutata *espressione_1* altrimenti si valuta *espressione_2*
- L'operatore condizionale implementa la valutazione pigra o lazy

Operatore condizionale

- L'operatore condizionale è simile all'istruzione **`if-else`**, tranne che *riporta il valore di un'espressione*

```
maggiore = (num1 > num2) ? num1 : num2;
           condizione      espr1  espr2
```

se `num1` è maggiore di `num2`, allora a `maggiore` viene assegnato `num1` altrimenti `num2`

- L'operatore condizionale è un operatore *ternario*, cioé richiede tre operandi

Operatore condizionale

```
System.out.println
  ("Il resto è di " + cont +
    (cont == 1) ? "euro" : "euri");
```

- se `cont` è 1, allora stampa `"euro"`. Per qualunque altro valore di `cont`, stampa `"euri"`

Precedenza tra operatori

- Gli operatori possono venire combinati in espressioni complesse

```
risultato = totale + cont / max - scarto;
```

- Gli operatori hanno una ***precedenza*** assegnata che determina l'ordine con cui vengono valutati
  - Moltiplicazione, divisione e resto sono valutati prima di somma, sottrazione e concatenazione tra stringhe
  - Le `()` per la valutazione dei parametri, l'operatore `dot` e gli operatori di auto-incr o auto-decr hanno massima precedenza
  - L'operatore di assegnamento la minima
  - Tabella precedenza operatori Java
- Gli operatori che hanno la stessa precedenza sono valutati da sinistra a destra
- Mediante le parentesi si può alterare l'ordine di precedenza

Precedenza tra operatori

- Ordine di valutazione dell'espressione:

```
a + b + c + d + e        a + b * c - d / e
1   2   3   4            2   1   4   3

a / (b + c) - d % e
2    1   4   3

z = a / (b * (c + (d - e)))
5   4   3   2   1
```


Associatività degli operatori

- Le regole di associatività stabiliscono l'ordine in cui eseguire le operazioni nel caso di due operatori nella stessa espressione che hanno la stessa precedenza
- In generale le operazioni possono essere eseguite da sinistra a destra (left-to-right), oppure da destra a sinistra (right-to-left)
- Tutti gli operatori binari in Java, assegnamenti esclusi, sono associativi a sinistra (left-to-right).
- Gli assegnamenti e gli operatori unari sono associativi a destra (right-to-left).

```
a = b += c = -d
```

viene valutata come

```
a = (b += (c = (-d)))
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;
          4        1   3   2
```

Poi il risultato è assegnato alla variabile alla sinistra

Assegnamento, ancora

- I membri di destra e sinistra dell'operatore **=** (**assegnamento**) possono contenere la stessa variabile

dapprima, si aggiunge 1 al valore originario della variabile

```
numero  =  numero + 1;
```

Poi il risultato è immagazzinato come nuove valore della variabile (il vecchio valore è sovrascritto)

Si può anche usare l'assegnamento `+=; numero += 1;`

Assegnamento in variabili di riferimento

- Si consideri l'assegnamento

```
x = y;
```

- Se `x` e `y` sono *variabili di tipo semplice*, il risultato dell'istruzione sarà che il contenuto di `y` viene copiato come contenuto di `x`
- Se `x` e `y` sono *variabili di tipo oggetto* e contengono il riferimento all'oggetto allora conseguenza dell'istruzione è che il riferimento di `y` viene copiato come contenuto di `x`
  - `x` e `y` si riferiscono ora allo stesso oggetto
- In entrambi i casi il contenuto della variabile `y` non cambia

Conversione tra tipi primitivi

- Qualche volta è utile o necessario convertire i dati da un tipo ad un altro
  - Un intero considerarlo come un numero in virgola mobile
- Le conversione vanno fatte con cautela per non perdere informazioni
- Le ***conversioni larghe*** sono sicure perchè vanno da un tipo di dato che viene rappresentato usando un certo quantitativo di memoria ad uno che ne usa di più (da `short` a `int`)
- Le ***conversioni strette*** possono perdere informazioni perchè restringono l'occupazione di memoria a dispozione del dato (da `int` a `short`)

Conversioni - 2

- Le conversioni tra tipi di dato possono avvenire in 3 modi:
  - a. ***Conversioni*** durante una operazione di assegnamento
  - b. ***Promozione*** in una espressione aritmetica
  - c. ***Casting***

a. quando un valore di un tipo viene assegnato ad una variabile di un altro tipo
  - È consentita solo la conversione larga

b. avviene automaticamente quando operatori aritmetici devono convertire gli operandi

Conversioni – 3

c. conversione detta ***casting*** è la tecnica di conversione più pericolosa e potente

- Tramite un casting esplicito si possono realizzare sia la *conversione larga* sia quella *stretta*
- Per effettuare un casting di tipo si dichiara il nuovo tipo tra `()` di fronte al valore di cui si vuole fare la conversione
- `int totale, numero;`
  `float risultato;` vogliamo dividere senza perdita d'informazione `totale` per `numero` effettuiamo un cast su `totale`:

```
risultato = (float) totale / numero;
```

Tabella conversioni tipi

da \ a	boolean	byte	short	char	int	long	float	double
boolean		n	n	n	n	n	n	n
byte 8bit	n		s	c	s	s	s	s
short 16bit	n	c		c	s	s	s	s
char 16 bit	n	c	c		s	s	s	s
int 32bit	n	c	c	c		s	s*	s
long 64bit	n	c	c	c	c		s*	s
float 32bit	n	c	c	c	c	c		s
double 64bit	n	c	c	c	c	c	c	

n non si applica; s viene fatto automaticamente, s* con perdita di precisione;
c mediante casting esplicito

Librerie di classi

- Una *libreria* è una collezione di classi che possono essere usate nei programmi
- La *libreria standard* fa parte di ogni sistema di sviluppo Java
- Le classi della libreria NON fanno parte del linguaggio, ma vengono usate continuamente
- La classe `System` e la classe `String` sono parte della libreria di classi standard di Java
- Altre librerie possono essere prodotte da terze parti o sviluppate da voi stessi

Packages

- Le classi della libreria standard sono organizzate in pacchetti

Package	*Scopo*
`java.lang`	supporto generale allo sviluppo importato automaticamente
`java.applet`	creare applets per il web
`java.awt`	grafica e interfacce grafiche
`javax.swing`	ulteriori componenti grafiche per GUI
`java.net`	comunicazione di rete
`java.util`	utilità varie
`java.text`	visualizzare testo formattato
`java.math`	eseguire calcoli

Dichiarazione di import

- Per usare una classe di un pacchetto, si indica il suo nome per esteso

```
java.util.Random;
```

- E la si importa

```
import java.util.Random;
```

- Per importare tutte le classi di un pacchetto si usa il carattere 'jolly' *

```
import java.util.*;
```


Dichiarazione di import - 2

- Tutte le classi del pacchetto `java.lang` sono importate automaticamente
  - Quindi non dobbiamo esplicitamente importare le classi `System` o `String` ad esempio
- La classe `Random` class parte del pacchetto `java.util` va importata
  - Fornisce metodi per generare numeri pseudocasuali
  - Numeri pseudocasuali sono numeri distribuiti senza regola apparente in un dato intervallo

Metodi statici di classe

- Alcuni metodi possono essere invocati tramite la classe, invece che tramite un oggetto della classe
- Sono i ***metodi di classe*** o ***metodi statici***
- La classe `Math` del pacchetto `java.lang` contiene molti metodi statici, che eseguono varie funzioni matematiche
  - Il valore assoluto, funzioni trigonometriche, radici quadrate, etc.

```
temp = Math.cos(90) + Math.sqrt(delta);
```

Invocazione dei metodi statici

- Si possono invocare in 2 modi:
  - `<NomeClasse>.<NomeMetodo>`
  - `<riferimentoOggetto>.<NomeMetodo>`
  - Usare sempre il **I** modo. Il **II** metodo è considerato improprio anche se possibile.
- I metodi statici sono risolti durante la compilazione
- I metodi d'istanza in esecuzione
- Le eventuali sottoclassi non hanno una propria copia dei metodi statici, ma condividono la stessa

Numeri pseudo-casuali

- La classe `Random` del pacchetto `java.util`, che va importato esplicitamente
- Per generare valori numerici pseudo-casuali è necessario istanziare un oggetto della classe: il *generatore di numeri casuali* con il costruttore `Random()`
- Quindi al generatore si possono richiedere servizi, invocando diversi metodi di generazione di numeri
- `nextInt() nextBoolean() nextFloat() nextInt(int n)` che restituisce un valore compreso tra `0` incluso e **`n`** escluso

Numeri pseudo-casuali

- I numeri pseudocasuali sono anche generabili con il metodo statico `random()` della classe `Math` del pacchetto `java.lang`

```
public static double random()
```

- Riporta un valore **`x`** nell'intervallo `0.0 <= x < 1.0`
- Per generare valori interi in un dato intervallo occorre effettuare un'operazione di scala e un cast esplicito.
  - Es: `0 <= x <= 4`

```
int x =(int) (Math.random() * 5)
```


Esempi

- `Echo.java` e `Quadratic.java`
- `TestDado.java` e `Dado.java`
- `RandomNumbers.java`