Informatica

Terzo anno

Prof. A. Longheu

Commenti

- Il commento sono informazioni ignorate dal compilatore, quindi nel commento si può scrivere qualsiasi cosa, anche in italiano; in genere si mette qualcosa di utile per capire il programma
- 2 modi

```
// commento fino alla fine della linea
/* C-style,
commento
su
più
linee */

// la variabile che segue serve per sommare i numeri
Int somma;
```

Keywords

- abstract
- default
- if
- private
- this
- boolean
- do
- implements
- protected
- throw
- extends
- null*
- break
- double
- import
- public
- throws
- byte
- else
- instanceof
- return
- transient
- case
- int
- false*
- short
- try
- catch
- final
- interface
- static
- void
- char
- finally
- long
- strictfp
- volatile
- true*
- class
- float
- native
- super
- while
- const
- for
- new
- switch
- continue
- goto
- package
- synchronized

false, true e null sono valori costanti (literals), non proprio parole chiave

Costanti e variabili

- I **dati** che si usano in un programma possono essere delle **costanti** o delle **variabili**:
  - Le prime possono essere immaginate come dei contenitori di una informazione stabilita una volta per tutte e non più modificabile
  - Le seconde come dei contenitori di una informazione che può essere modificata nel corso del programma

Identificatori

- Gli identificatori sono nomi di variabili e/o costanti
- Non possono essere gli stessi delle keyword
- Sono case-sensitive
- Possono avere qualsiasi lunghezza
- Si possono comporre utilizzando i seguenti caratteri:
  - Lettere e cifre in Unicode, per cui שקא, БЛЪЙЧ e ٣شـج sono tutti identificatori validi; utilizzando l'unicode, alcuni caratteri sono graficamente simili se non uguali, anche se hanno codici unicode differenti; è quindi consigliabile utilizzare un alfabeto soltanto nel sorgente
  - Il carattere underscore (_) e dollaro ($)
  - Devono iniziare con una lettera

Tipi di dato

- Come in tutti i linguaggi di programmazione a ciascuna variabile e costante è associato anche il **TIPO**, ovvero la classe di valori che la costante o variabile può assumere nel corso dell'esecuzione del programma (e quindi gli operatori applicabili al valore in essa contenuto).

- L'associazione di un nome (di costante o di variabile) ad un tipo di dato non cambia mai durante l'esecuzione del programma.

Tipi di dato

- Il concetto di *tipo di dato* viene introdotto quindi per raggiungere due obiettivi:
  - esprimere in modo sintetico
    - la rappresentazione dei dati in memoria
    - un insieme di operazioni ammissibili
  - permettere di *effettuare controlli statici* (al momento della compilazione) sulla ***correttezza*** del programma

Tipi di dato

- Quali sono i tipi di dato ammessi in Java?
- Ogni entità che ha un valore (variabile, parametro, valore restituito da un metodo) deve avere un tipo.
- In Java ci sono due generi di tipi :
  - Tipi primitivi
  - Tipi riferimento

Tipi primitivi

- char
- byte
- short
- int
- long
- boolean
- float
- double

Tipi primitivi

Data Type	Values	Storage (in bytes)
`char`	0 to 65535	2 (16 bits)
`byte`	–128 to 127	1 (8 bits)
`short`	–32768 to 32767	2 (16 bits)
`int`	–2147483648 to 2147483647	4 (32 bits)
`long`	–9223372036854775808 to 9223372036854775807	8 (64 bits)

- I tipi utilizzano tutti il segno ed operano con il complemento a due

Tipo int

- Il tipo "**int**" è ben diverso dal tipo INTERO inteso in senso matematico dove
  INTERO **Z** $\{-\infty,\ldots,-2,-1,0,+1,+2,\ldots,+\infty\}$

- Ovvero il tipo "**int**" ha un insieme di valori e di operazioni definibili su di essi limitato a priori:
  - L'insieme dei valori dipende dalla macchina, in Java 4 byte

- Questo principio vale per tutti i tipi numerici quindi si devono sempre fare i conti con i limiti della rappresentazione (possibile overflow o underflow)

Tipi primitivi

- Boolean
  - boolean (1 bit) false e true
  - tipo autonomo totalmente disaccoppiato dagli interi: non si convertono boolean in interi e viceversa, neanche con un cast
  - le espressioni relazionali e logiche danno come risultato un boolean, non un int come in C
- float e double rappresentano numeri in virgola mobile secondo lo standard IEEE754, con mantissa ed esponente

Operatori per il tipo int

- Al tipo **int** (e tipi ottenuti da questo mediante qualificazione) sono applicabili i seguenti operatori:

Operatori aritmetici	**Risultato**
+ - * /	int
% (resto divisione intera)	int

Operatori relazionali	**Risultato**
== !=	(0 se falso, ≠ 0 se vero)
< > <= >=	(0 se falso, ≠ 0 se vero)

Operatori per il tipo int

- + Addizione
- - Sottrazione
- * Moltiplicazione
- / Divisione intera Es., 5/3=1
- % Modulo (resto dalla divisione intera) Es., 5%3 = 2
- == Operatore relazionale di uguaglianza *(ATT.: diverso dal simbolo = che denota l'operazione di assegnamento!)*
- != Operatore relazionale di diversità
- > Operatore relazionale di maggiore stretto
- < Operatore relazionale di minore stretto
- >= Operatore relazionale di maggiore-uguale
- <= Operatore relazionale di minore-uguale

Operatori per i tipi float e double

Operatori aritmetici	Risultato
+ - * /	float o double

Operatori relazionali	Risultato
== !=	(0 se falso, ≠ 0 se vero)
< > <= >=	(0 se falso, ≠ 0 se vero)

Operatori per i tipi float e double

- + Addizione
- - Sottrazione
- * Moltiplicazione
- / Divisione reale
  (*anche se si usa lo stesso simbolo, è diversa da quella intera*)
- == Operatore relazionale di uguaglianza
  (ATT.: diverso dal simbolo = che denota l'assegnamento!)
- != Operatore relazionale di diversità
- > Operatore relazionale di maggiore stretto
- < Operatore relazionale di minore stretto
- >= Operatore relazionale di maggiore-uguale
- <= Operatore relazionale di minore-uguale

Attenzione!

- A causa della rappresentazione su di un numero finito di cifre, ci possono essere errori dovuti al troncamento o all'arrotondamento di alcune cifre decimali
- Meglio evitare l'uso dell'operatore **==**
- **I test di uguaglianza tra valori reali (in teoria uguali) potrebbero non essere verificati.**
- Es: se x e y sono due variabili float, allora non è detto che risulti verificata la seguente **(x / y) *y == x**

- Meglio utilizzare "**un margine accettabile di errore**":
  (x == y) → (x <= y+epsilon) && (x >= y-epsilon)
  (dove ad esempio si è posto: epsilon=0.000001)

Costanti letterali

- Le **costanti letterali (literals)** sono valori costanti utilizzati in un programma.
- Esempi:

```
x = y + 3;
System.out.println("Ciao");
finished = false;
```

Costanti letterali

- I **caratteri** sono codificati utilizzando il codice unicode:
  - i caratteri singoli sono rappresentati come in C/C++ racchiusi da apici singoli, le stringhe invece richiedono le virgolette

    'a' 'b' 'C' '\n' '\t' "ciao" "prova"
  - Qualsiasi cosa che inizia con \u è trattata come il valore corrispondente al numero unicode

    '\u03df' '\u003f'

Costanti letterali

- Per i literals **numerici interi** (byte, short, int, long) di default si usano i numeri in base 10

  23 100 0 1234567

- Java supporta anche i numeri in base 16…

  0x23 0x1 0xaf3c21 0xAF3C21

- …E in base base 8 (preceduto da 0, quindi un numero che inizia con 0 è automaticamente considerato ottale)

  023 01 0123724

- Se il literal ha il suffisso '**l**' o '**L**', è di tipo long (si usa di solito il maiuscolo per non confondere 'l' con '1')

  23L 100l 0xABCDEF01234L

- senza la 'l' la costante è normalmente considerata int (anche se il numero fosse abbastanza piccolo da entrare in un byte o short)
- i **literal per il boolean** sono solo true e false

Costanti letterali

Per i **literals numerici in virgola mobile** (float e double) i valori sono quelli ammessi dallo standard IEEE-754:

- float (4 byte) 10^{-45} ... + 10^{38}
- double (8 byte) 10^{-328} ... + 10^{308}
- Le costanti float terminano con la lettera F
  - 3.54 è una double (è possibile utilizzare l'estensione D o d)
  - 3.54F è una costante float
  - double x = 3.54; double x = 3.54F; float fx = 3.54F; sono OK
  - float fx = 3.54; è una frase illecita
- sono presenti +0.0 e -0.0, uguali per l'operatore == ma che possono fare ottenere risultati diversi quando nei calcoli

Dichiarazione di costanti

- Una dichiarazione di **costante** associa permanentemente un valore ad un identificatore
- Si attribuisce ad ogni costante un **tipo**
- Si utilizza la parola riservata **final**

- Sintassi:

  Final <tipo> <identificatore> **=** <espr> ;

- Esempio
  - `final int N = 100;`
  - `final float pigreco = 3.1415;`
  - `final char sim = 'A';`

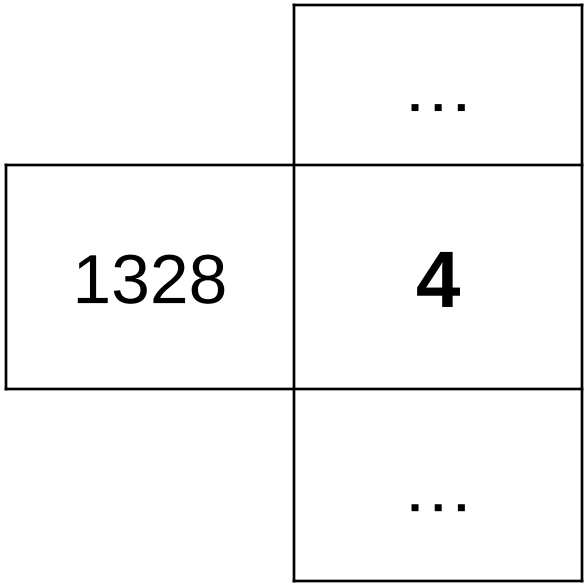
Variabili

- Una **variabile** è un'astrazione della *cella di memoria*
- Formalmente, è un simbolo *associato a un indirizzo fisico di memoria*

simbolo	***indirizzo***
x	1328

Variabili

- … che contiene un valore:



- Il valore di x è *attualmente* 4 (può cambiare!), l'indirizzo cambia al massimo fra un'esecuzione e la successiva dello stesso programma, ma non lo vediamo

Dichiarazione di variabili

- Una variabile utilizzata in un programma deve essere dichiarata
- La dichiarazione può essere fatta ovunque ma in ogni caso prima dell'utilizzo della variabile; spesso si mettono tutte all'inizio del programma per leggibilita' (ma non è necessario)

- La dichiarazione è composta da
  - il nome della variabile (identificatore)
  - il *tipo* dei valori che possono essere denotati alla variabile

Dichiarazione di variabili

- Scopo:
  - Elencare tutte le variabili che saranno utilizzate nella parte esecutiva
  - Attribuire ad ogni variabile un tipo
  - È possibile raggruppare le dichiarazioni di più variabili dello stesso tipo in una lista separata da virgole

Dichiarazione di variabili

- Sintassi:
  - <tipo> <identificatore>;

- Esempi:
  - int x; /* x deve denotare un valore intero */
  - float y; /* y deve denotare un valore reale */
  - char ch; /* ch deve denotare un carattere */
  - int a, b, c;
  - float z, w;

Dichiarazione di variabili

- Sintassi:

  <tipo> <elenco_di_variabili>;

  <elenco_di_variabili> ::= <ident>{,<ident>}

- Esempi:
  - int x,z; /* x e z devono denotare un valore intero */
  - float y,w; /* y e w devono denotare un valore reale */

Inizializzazione di variabili

- Contestualmente alla *definizione* è possibile *specificare un valore iniziale* per una variabile

- Inizializzazione di una variabile:

  <tipo> <identificatore> = <espr> ;

- Esempio
  - int x = 32;
  - double speed = 124.6;

Espressioni

- Una *espressione* è una notazione che denota un valore mediante un *processo di valutazione*
- Una espressione può essere *semplice* o *composta* (tramite aggregazione di altre espressioni)

Espressioni semplici

Quali espressioni elementari?

- costanti

  ‘A’ 23.4 -3 “ciao” ....

- simboli di variabile

  x pippo pigreco ….

- simboli di funzione

  f(x)
  concat(“alfa”,”beta”)
  …

Operatori ed espressioni composte

- Ogni linguaggio introduce un **insieme di *operatori***
- … che permettono di ***aggregare altre espressioni (operandi)***
- … per formare ***espressioni composte***
- … con riferimento a diversi **domini / tipi di dato** (numeri, testi, ecc.)

- Esempi
  - 2 + f(x)
  - 4 * 8 - 3 % 2 + arcsin(0.5)
  - BufferedReader(InputStreamReader(System.in))
  - a && (b || c)
  - …

Classificazione degli operatori

- Due criteri di classificazione:
  - in base al *tipo* degli operandi
  - in base al *numero* degli operandi

In base al tipo degli operandi	In base al numero degli operandi
• aritmetici • relazionali • logici • condizionale	• unari • binari • ternari • …

Operatori aritmetici

Operazione	Operatore	Simbolo
Inversione di segno	unario	-
Somma	binario	+
Differenza	binario	-
Moltiplicazione	binario	*
Divisione fra interi	binario	/
Divisione fra reali	binario	/
Modulo (fra interi)	binario	%

NB: la divisione a/b è fra interi se sia a che b sono interi, è fra reali in tutti gli altri casi

Operatori: overloading

- In tutti i linguaggi, operazioni primitive associate a tipi diversi possono essere denotate con lo stesso simbolo. Ad esempio, le operazioni aritmetiche su reali o interi
- In realtà l'operazione è diversa e può produrre risultati diversi

```
int X, Y;
Se X=10 e Y=4;
X/Y vale 2
```

```
int X, float Y;
Se X=10 e Y=4.0;
X/Y vale 2.5
```

```
float X, Y;
Se X=10.0 e Y=4.0;
X/Y vale 2.5
```

Conversioni di tipo

- E’ possibile combinare tra di loro operandi di tipo diverso:
  - espressioni **omogenee**: tutti gli operandi sono dello stesso tipo
  - espressioni **eterogenee**: gli operandi sono di tipi diversi

Regola adottata :

- sono eseguibili le espressioni eterogenee in cui tutti i tipi referenziati risultano ***compatibili*** (cioè è possibile convertirli implicitamente in modo che tutti risultino omogenei)

Compatibilità di tipo

- 3 / 4.2 è una divisione *fra reali,* in cui il primo operando è convertito automaticamente da **int** a **double (da 3 a 3.0)**

- **3 % 4.2** è una operazione *non ammissibile*, perché 4.2 non può essere convertito in **int (l'operatore % infatti è valido solo per gli interi)**

Conversioni di tipo

Data una espressione ***x op y***

1. Se l'espressione è eterogenea, rispetto alla seguente gerarchia

Byte < short < int < long < float < double

si converte temporaneamente l'operando di tipo *inferiore* al tipo *superiore* (***promotion***)

2. A questo punto l'espressione è **omogenea** e viene eseguita l'operazione specificata. Il risultato è di tipo uguale a quello prodotto dall'operatore effettivamente eseguito (in caso di overloading, quello più alto gerarchicamente)

Conversioni di tipo - esempio

```
int x;
int y;
double r;
(x+y) / r
```

La valutazione dell'espressione procede da sinistra verso destra

Passo 1: **(x+y)**

- viene applicata la somma tra interi
- **risultato intero *tmp***

Passo 2 (***tmp*** **/ r)**

- ***tmp*** viene convertito nel double corrispondente
- viene applicata la divisione tra reali
- **risultato reale**

Compatibilità di tipo

- In un assegnamento, l'identificatore di variabile e l'espressione devono essere dello stesso tipo
- Nel caso di tipi diversi:
  - se possibile si effettua la conversione implicita,
  - altrimenti l'assegnamento può generare perdita di informazione

```
int x;
double r;

r = x;     /* int -> double*/
x = r;     /* troncamento!!! */
```

Compatibilità in assegnamento

- In generale, sono automatiche le conversioni di tipo che non provocano perdita d'informazione
- Tuttavia, le espressioni che *possono* provocare perdita di informazioni non sono illegali

Esempio

```
int i=5; float f=2.71F;; double d=3.1415;
f = f+i;          /* int convertito in float */
i = d/f;          /* double convertito in int !*/
f = d;   /* arrotondamento o troncamento */
```

Possibile **warning**: la conversione può causare la perdita di bits significativi

Casting

- In qualunque espressione è possibile forzare una particolare conversione utilizzando l'**operatore di cast**

Sintassi

(<tipo> **)** <espressione>

Esempi

```
int y;
int z;
float x;
.....
x = (float)somma/numero;
```

Operatori relazionali

- Sono tutti operatori binari:

Relazione	simbolo
Uguaglianza	==
Diversità	!=
Maggiore di	>
Minore di	<
Maggiore o uguale a	>=
Minore o uguale a	<=

Operatori logici

connettivo logico	Operatore	simbolo
not (negazione)	unario	!
and	binario	&&
or	binario	||

Operatori logici

- Anche qui sono possibili espressioni miste, utili in casi specifici

```
5 && 7          0 || 33          !5
```

- **Valutazione in *corto-circuito***
  - la valutazione dell'espressione cessa *appena si è in grado di determinare il risultato*
  - il secondo operando è valutato *solo se necessario*

Valutazione in corto circuito

- `22 || x`

  già vera in partenza perché 22 è vero

- `0 && x`

  già falsa in partenza perché 0 è falso

- `a && b && c`

  se a&&b è falso, il secondo && non viene neanche valutato

- `a || b || c`

  se a||b è vero, il secondo || non viene neanche valutato

Priorità degli operatori

- **PRIORITÀ: specifica l'ordine di valutazione degli operatori quando in una espressione compaiono *operatori diversi***

- Esempio: `3 + 10 * 20`
  - si legge come 3 + (10 * 20) perché l'operatore * è più prioritario di +

- NB: operatori diversi possono comunque avere *egual priorità*

Associatività degli operatori

- **ASSOCIATIVITÀ**: **specifica l'ordine di valutazione degli operatori quando in una espressione compaiono *operatori di egual priorità***

- Un operatore può quindi essere *associativo a sinistra* o *associativo a destra*

- Esempio: `3 - 10 + 8`
  - si legge come (3 - 10) + 8 perché gli operatori - e + sono equiprioritari e **associativi a sinistra**

Priorità e associatività

- Priorità e associatività predefinite possono essere alterate mediante *l'uso di **parentesi** (tonde)*
- Esempio: `(3 + 10) * 20`
  - denota 260 (anziché 203)
- Esempio: `30 - (10 + 8)`
  - denota 12 (anziché 28)

Priorità e associatività

- Gli operatori relazionali hanno **priorità inferiore** agli operatori aritmetici

k < b+3

- equivale a k < (b+3)
- e non a (k<b) + 3

Dove si sbaglia frequentemente…

Operazioni matematiche e tipi di dato

- Divisione tra interi e divisione tra reali (stesso simbolo /, ma significato differente)
- Significato e uso dell'operatore di modulo (%)
- Operatore di assegnamento (=) e operatore di uguaglianza (==)