Lab. Programmazione (CdL Informatica) & Informatica (CdL Matematica) a.a. 2022-23

Monica Nesi

Università degli Studi dell'Aquila

18 Ottobre 2022

Tipi e Dichiarazioni

Nelle classi viste finora abbiamo scritto righe di codice del tipo:

```
int x = Integer.parseInt(args[0]);
double a = Double.parseDouble(args[1]);
```

Questi sono esempi di *dichiarazioni di identificatori di variabili* di *tipo* `int` e `double`, rispettivamente, con *inizializzazione* al valore risultante dalla valutazione dell'espressione a destra del simbolo =.

Con tali dichiarazioni si afferma di voler utilizzare due locazioni di memoria, riferite con i nomi (identificatori) `x` ed `a`, atte a contenere rispettivamente valori interi e valori decimali, il cui contenuto può variare (*variabili*), inizializzate con il valore dell'espressione a destra corrispondente.

Dichiarazioni

N.B. In Java qualunque identificatore di variabile deve essere dichiarato ed inizializzato *prima* di essere utilizzato.

Il tipo viene dichiarato *una volta sola*.

Le dichiarazioni viste sono istanze della seguente *sintassi* per le dichiarazioni di (identificatori di) variabili:

$$<Tipo> <Ide> = <Espr> ;$$

dove $<Tipo>$ denota l'insieme (*categoria sintattica*) dei tipi del linguaggio, $<Ide>$ è l'insieme degli identificatori ed $<Espr>$ denota l'insieme delle espressioni del linguaggio.

Una condizione non espressa dalla sintassi delle dichiarazioni è che l'espressione a destra del simbolo $=$ deve avere un tipo uguale al tipo a sinistra (o almeno *compatibile*).

Dichiarazioni (cont.)

È possibile dichiarare un identificatore senza inizializzazione secondo la sintassi:

$$<\textit{Tipo}> <\textit{Ide}> ;$$

come in

```
int x;
double a;
```

Però, prima di utilizzare tali variabili, occorrerà inizializzarle, e.g.

```
x = 7;
a = 4.6;
```

secondo la sintassi del comando di *assegnamento*

$$<\textit{Ide}> = <\textit{Espr}> ;$$

N.B. Non si ripete il tipo dell'identificatore.

Dichiarazioni multiple

La dichiarazione di più variabili *tutte dello stesso tipo* può essere fatta in modo compatto tramite una dichiarazione multipla:

```
int x;
int y;
```

si può scrivere anche

```
int x,y;
```

Le due dichiarazioni con inizializzazione

```
int x = 3;
int y = -2;
```

possono essere scritte anche come

```
int x = 3, y = -2;
```

Dichiarazioni multiple (cont.)

Notare che in

```
int x = 1, y;
```

la variabile `y` non è inizializzata.
Se cerco di stampare il suo valore con

```
System.out.println(y);
```

si ha errore in fase di compilazione con un messaggio del tipo "variable y might not have been initialized".

Lo stesso vale per

```
int x, y = 2;
```

dove `x` non è inizializzata.

Tipi di dato base o primitivi

Il nome di una variabile la identifica ed il suo tipo definisce la sua dimensione e l'insieme di operazioni che vi si possono eseguire.

I tipi `int` e `double` sono due dei *tipi base o primitivi* del linguaggio Java. Questi tipi sono:

- logico o booleano `boolean`;
- numerici interi `byte`, `short`, `int` e `long`;
- numerici decimali (con virgola) `float` e `double`;
- caratteri `char`.

N.B. Il nome di questi tipi comincia con lettera minuscola.

Tipo boolean

Il tipo `boolean` per i booleani contiene i valori costanti `true` e `false`.

N.B. Questi valori sono diversi da 1 e 0, che in altri linguaggi vengono usati per rappresentare i valori vero e falso.

Il risultato di confronti tra valori numerici è di tipo `boolean`.

Il seguente frammento di programma

```
boolean p = true, q = 3>7;
System.out.println(p);
System.out.println(q);
```

dà l'output:

```
true
false
```

Tipi per i numeri interi

Abbiamo 4 tipi con diverse *capacità*, i.e. numeri di bit per rappresentare i numeri interi.

Si usa la *rappresentazione in complemento a 2*: con n bit si rappresentano i numeri nell'intervallo $[-2^{n-1}, 2^{n-1}-1]$.

- `byte` intero su 8 bit $[-128, 127]$
- `short` intero su 16 bit $[-2^{15}, 2^{15}-1]$
- `int` intero su 32 bit $[-2^{31}, 2^{31}-1]$
- `long` intero su 64 bit $[-2^{63}, 2^{63}-1]$

Tipicamente noi useremo il tipo `int`.

Tipi per i numeri decimali

Abbiamo 2 tipi con capacità diverse.

Si usa la *rappresentazione in virgola mobile (floating point)*.

- `float` su 32 bit (8 cifre significative)
- `double` su 64 bit (16 cifre significative, consente una maggiore precisione)

Tipicamente noi useremo il tipo `double`.

Conversioni di tipo

Se una variabile di un dato tipo viene assegnata ad una variabile di un tipo *più capace*, si ha una conversione automatica, detta *promozione*:

```
byte b = 121;
int i = b;
double d = i;
```

L'operazione contraria, i.e. l'assegnamento di una variabile di un dato tipo ad una variabile di un tipo *meno capace*, non è automatica (si potrebbe avere perdita di cifre significative):

```
int i = 50;
byte b = i;
```

Si ha un errore in fase di compilazione con un messaggio del tipo "incompatible types: possible lossy conversion from int to byte"

Ma il valore 50 può essere assegnato alla variabile b!

Cast

Occorre un *cast*, ovvero occorre *forzare* il tipo di una variabile ad un tipo diverso.

```
int i = 50;
byte b = (byte)i;
```

Invece se abbiamo:

```
int i = 257;
byte b = (byte)i;
```

si ha che `b` vale `1`. Qui si ha perdita di cifre significative.

```
double d = 7.5;
int i = (int)d;
```

La variabile `i` vale `7`, i.e. la parte intera del valore decimale `7.5`.

Cast (cont.)

Se abbiamo

```
byte b1 = 1, b2 = 2;
byte b3 = b1+b2;
```

Si ha un errore in fase di compilazione con un messaggio del tipo "incompatible types: possible lossy conversion from int to byte"

Il problema è nella somma `b1+b2`, in quanto i risultati parziali del calcolo di espressioni vengono memorizzati in variabili implicite di tipo `int` (a meno che non vi siano variabili di tipo più capace nelle espressioni).

Occorre un cast:

```
byte b3 = (byte)(b1+b2);
```

Tipo char

Il tipo `char` per i caratteri alfanumerici li rappresenta su 16 bit come interi non negativi nell'intervallo $[0, 65535]$.

Java si basa sullo standard di codifica Unicode, che estende il codice ASCII su 8 bit (i primi 128 caratteri di Unicode corrispondono all'insieme di caratteri ASCII).

In Java una costante di tipo `char` è un carattere racchiuso tra singoli apici.

Il seguente frammento di programma

```
char c = '3', d = 'c';
System.out.println(c);
System.out.println(d);
```

dà l'output:

```
3
c
```

Le costanti

In Java esistono *costanti letterali* (*literal*) e *costanti simboliche*.

Le costanti letterali rappresentano esplicitamente i valori di un determinato tipo.

Esempi: 12 è una costante letterale di tipo `int`, `'t'` è una costante letterale di tipo `char`, 3.7 è una costante letterale di tipo `double`.

Le costanti simboliche sono *nomi usati per denotare valori costanti*, i.e. che non possono essere modificati. Esempio: il valore π.

Le costanti simboliche si distinguono dalle variabili perché nella dichiarazione viene aggiunta la parola riservata `final`.

Si assegna il valore alla costante in fase di dichiarazione oppure successivamente, ma si può fare *un solo* assegnamento.

Costanti simboliche

```
final double PI = 3.14;
```

oppure

```
final double PI;
PI = 3.14;
```

Convenzione standard di Java: il nome di una costante simbolica contiene tutte lettere maiuscole. In caso di più parole nel nome, queste sono separate da un trattino basso (underscore).

```
final int DAYS_IN_WEEK = 7;
```

Le costanti possono rendere un programma più leggibile.

Costanti simboliche: esempio

```
class Volume {
  public static void main (String[] args) {
    int b = 2;       // numero bottiglie
    int l = 10;      // numero lattine

    /* calcola il volume totale con bottiglie
       da 2 litri e lattine da 0.25 litri */

    double totale = b*2 + l*0.25;

    /* stampa il risultato */
    System.out.println("Il volume totale e' "
    + totale + " litri.");
  }
}
```

Costanti simboliche: esempio (cont.)

Pessimo stile: esistono più occorrenze del valore 2 che denotano concetti diversi.

Il programma risulta più leggibile se ai valori costanti (i.e. le capacità di bottiglie e lattine) vengono dati nomi simbolici.

Vengono definite due costanti:

```
final double BVOLUME = 2;          // 2.0
final double LVOLUME = 0.25;
```

e poi il totale viene calcolato come segue:

```
double totale = b*BVOLUME + l*LVOLUME;
```

Costanti simboliche: esempio (cont.)

Il programma è anche più facile da modificare.

Supponiamo di avere bottiglie da 1.5 litri invece che 2 litri.

Invece di cercare ogni occorrenza di 2 e determinare se va rimpiazzata con 1.5, basta cambiare il valore assegnato alla costante `BVOLUME`, salvare la modifica e ricompilare.

Lo stesso discorso vale per la costante `LVOLUME`.

Quindi si fa l'*aggiornamento solo in un punto del programma*.

Input diversi: dare il numero di bottiglie e lattine in input.

Costanti simboliche: esempio (cont.)

```
class Volume {
  public static void main (String[] args) {
    final double BVOLUME = 2.0;
    final double LVOLUME = 0.25;
    int b = Integer.parseInt(args[0]);
    int l = Integer.parseInt(args[1]);

    /* calcola il volume totale */
    double totale = b*BVOLUME + l*LVOLUME;

    /* stampa il risultato */
    System.out.println("Il volume totale e' "
    + totale + " litri.");
  }
}
```

Costanti simboliche: esempio (cont.)

```
javac Volume.java
java Volume 2 10
Il volume totale e' 6.5 litri.
```

Cose da ricordare!

Dichiarare ed inizializzare una variabile *prima* di utilizzarla.

Convenzione standard su costanti e loro nome.

Commenti in Java

- Tutto ciò che segue // fino alla fine della riga è interpretato come commento.

- Tutto ciò che è racchiuso tra /* e */, su una singola riga o su righe successive, è interpretato come commento.