Azzolini Riccardo 2018-10-09

Tipi, variabili e metodi statici

1 Linguaggio staticamente tipizzato

Java è un linguaggio **staticamente** (detto anche *fortemente*) **tipizzato** (o *tipato*) perché il tipo di ogni variabile ed espressione presente nel programma è noto *al momento della compilazione*.

Il compilatore effettua quindi i controlli di compatibilità dei tipi, che aiutano a evitare errori.

2 Tipo delle espressioni

Il tipo di un'espressione dipende dall'operatore e dai tipi degli operandi.

2.1 Esempio

Espressione `x + y`:

- se `int x; int y;`, allora:
  - `+` denota la somma tra numeri interi
  - il risultato è di tipo `int`
  - in fase di esecuzione, la **valutazione** avviene sommando i numeri contenuti in x e y
- se `String x; String y`, allora:
  - `+` denota la concatenazione di stringhe
  - il risultato è di tipo `String`
  - la valutazione avviene costruendo una nuova stringa corrispondente alla concatenazione delle stringhe a cui fanno riferimento `x` e `y`
- se `String x; int y`, allora:
  - `+` denota la concatenazione di stringhe
  - il risultato è di tipo `String`

– la valutazione avviene concatenando la stringa a cui fa riferimento `x` con una che rappresenta il numero intero contenuto in `y` (**conversione implicita**)

3 Dichiarazione e definizione di variabili

Per **dichiarare** una variabile è necessario scriverne il tipo, seguito dal nome. È anche possibile dichiarare su una singola riga più variabili dello stesso tipo, scrivendo più nomi separati da virgola.

```
Tipo var1, var2, ...;
```

È possibile **inizializzare** una variabile, cioè assegnarle un valore, insieme alla dichiarazione: questa forma abbreviata si chiama **definizione** di una variabile.

```
Tipo var1 = espr1, var2 = espr2, ...;
```

4 Tipi primitivi e tipi riferimento

In Java, i tipi sono suddivisi in due categorie: **tipi primitivi** e **tipi riferimento**.

4.1 Tipi primitivi

Le variabili di tipo primitivi contengono *direttamente* un valore.

I tipi primitivi, tutti predefiniti (non è possibile definirne altri), sono:

numeri interi: `byte`, `short`, `int` e `long` (in ordine crescente di capacità)

numeri a virgola mobile: `float` e `double`

caratteri: `char`

booleani: `boolean`

Questi tipi esistono per motivi di efficienza, ma introducono alcune complicazioni nella programmazione.

4.2 Tipi riferimento

Le variabili di tipo riferimento contengono un *riferimento* a un oggetto, cioè un'informazione che permette di accedere a tale oggetto.

Alcuni tipi riferimento (es. `String`) sono predefiniti, molti altri sono definiti in librerie (standard e non), ed è possibile definirne altri ancora nei propri programmi.

In generale, i tipi riferimento si suddividono in tre categorie:

- classi
- interfacce
- array

4.3 Assegnamento

A prescindere dal tipo, l'assegnamento tra variabili (es. `v = u;`) esegue sempre una copia del *contenuto* di una variabile (`u`) in un'altra (`v`).

- Tra variabili di tipo primitivo, l'assegnamento crea quindi una *copia del valore*. Ad esempio:

  ```
  int x = 10;
  int y;

  y = x; // y contiene una copia del numero 10
  ```

- Tra variabili di tipo riferimento, invece, l'assegnamento crea una *copia del riferimento*: dopo l'assegnamento, entrambe le variabili faranno riferimento allo stesso oggetto. Ad esempio:

  ```
  String x = "pippo";
  String y;

  y = x; // y contiene un secondo riferimento alla stessa stringa "pippo"
  ```

  Di conseguenza, per i tipi riferimento mutabili, le modifiche effettuate su un oggetto tramite una specifica variabile saranno visibili anche da tutte le altre variabili che fanno riferimento allo stesso oggetto.

4.4 Letterale `null`

`null` rappresenta un valore assegnabile a tutte le variabili di tipo riferimento. Inoltre, è il valore di default di tali variabili (se non vengono inizializzate).

Per convenzione, denota l'assenza di un riferimento a un oggetto (o un "riferimento vuoto"). Un tentativo di accesso a un oggetto tramite un riferimento `null` provoca un errore di esecuzione.

5 Metodi statici

Sono servizi forniti *direttamente dalle classi*, anziché dai singoli oggetti.

Invocazione:

```
nome_classe.nome_metodo(lista_argomenti)
```

Alcuni degli impieghi più comuni sono:

- costruire oggetti della classe stessa da oggetti o valori di un altro tipo
- fornire operazioni utili su oggetti o su tipi primitivi
- definire proprietà che influenzano tutti gli oggetti di una classe

5.1 Esempi

- classe `java.lang.String`:
  - `public static String valueOf(int i)`: restituisce un riferimento a una stringa che rappresenta il valore di `i`
- classe `java.lang.Math`:
  - `public static double cos(double a)`
  - `public static double log(double a)`
  - `public static double log10(double a)`
- classe `java.lang.Integer` (classe corrispondente al tipo primitivo `int`):
  - `public static int parseInt(String s)`: restituisce il valore intero corrispondente alle cifre contenute nella stringa a cui si riferisce `s`, causando un errore di esecuzione se invece la stringa non rappresenta un numero intero
  - `public static String toBinaryString(int i)`: restituisce un riferimento a una stringa contenente la rappresentazione binaria del numero `i`