# Spieghiamo un po’ di nomenclatura Java.

Le variabili possono essere chiamate genericamente con il nome di variabile possono assumere dei nomi particolari a seconda della posizione in cui sono dichiarate.

Per esempio vediamo di seguito che:

public class nome\_classe{

float nome\_variabile1

La variabile1 appena dichiarata in precedenza può essere chiamata campo della classe o variabile membro o variabile globale questo perchè ha una vità molto duratura. Infatti vivrà e rimarra memorizzata fino a che l'istanza della classe di cui fa parte esisterà.

Questo che definiamo qui sotto si chiama metodo o funzione

public static void nome\_metodo(float nome\_variabile2){

float nome\_variabile3

Sia nome\_variabile2 e nome\_variabile3 avranno una vità invece più ristretta perchè esisteranno solo all'interno del metodo che le contiene.

Però se nome\_variabile3 viene chiamata variabile locale, diversamente nome\_variabile2 si chiamerà parametro in ingresso.

}

public static void main(String[] args){

Scanner in = new Scanner(System.in);

Con la riga appena scritta abbiamo definito quella che si chiama un istanza o un oggetto della classe Scanner.

Questa ha la stessa forma di una qualsiasi variabile che noi possiamo dichiarare ma siccome non è una variabile di tipo primitivo (cioè non è ne un intero, un numero a virgola mobile o un booleano) allora si dice che è un istanza di una classe.

Questo perchè solitamente una classe è molto più complessa di un tipo di dato primitivo.

In teoria anche le variabili di tipo String dovrebbero essere chiamate istanze della classe String, ma visto che vengono usate cosi in modo frequente e semplice si è preso l'abitudine di chiamarle variabili e considerare String un tipo di dato primitivo, ma in realtà non lo è. Questo perchè String è presente in Java già dalla versione 1.0, però non esiste in linguaggi da cui Java deriva come C.

}

}

# Spieghiamo un po di regole per la definizione di metodi in modo corretto.

Sappiamo che un metodo si dichiara in un certo modo che è il seguente:

<modificatori> <tipo> <nome> (<lista\_parametri\_formali>) {

<corpo>

}

- modificatori: o attributi indicano di che tipo di metodo si parla.

- tipo: indica che tipo di dato il metodo restituisca al richiamante.

- nome: è il nome del metodo che si sta dichiarando.

- lista parametri formali: sono tutti i tipi e nomi di variabili che il metodo riceverà dal richiamante

Quando si parla di metodi è importante il concetto di firma del metodo perchè questo ci permette di evitare ambiguita nella dichiarazione di metodi.Infatti non possono esistere due metodi con la stessa firma.

Se abbiamo un generico metodo definito come di seguito

<modificatori> <tipo> <nome> (<tipo1> <p1> , ... , <tipon> <pn>) {

...

}

la firma corrispondente è data dalla sequenza:

(<nome>, <tipo1>, ..., <tipon>)

Quindi quando si scrive un metodo bisogna stare bene attenti a non scrivere

metodi che abbiano la stessa firma di un altro metodo che abbiamo già scritto

o che fa parte già delle librerie di Java.

# Static or not static? That's the problem

Per capire in che caso ci fa comodo definire un metodo static o definirlo non static bisogna vedere questo argomento da due punti di vista opposti:

* Punto di vista della classe che contiene i metodi
* Punto di vista della classe dove vengono richiamati i metodi static o non static (tipicamente il punto di vista coincide con quello dal main)

Punto di vista della classe contenitrice

Se abbiamo una classe:

public class Lib\_Funzioni{

int time;

public void setTime(int t){

time=t;

}

public static void setTime2(int t){

time=t;

}

}

La caratteristica principale per distinguere i metodi statici da quelli non statici è che i metodi statici non possono accedere alle variabili contenute nella propria classe.

Cioè nell’esempio qui sopra setTime2 che è un metodo static non può accedere alla variabile time per cui il compilatore ci darà un errore.

Questo aspetto ci farebbe pensare che i metodi static siano limitanti e scomodi, in realtà hanno un risvolto positivo.

Punto di vista della classe richiamante

Se all’interno del nostro programma dobbiamo andare a richiamare un metodo static o uno non static la differenza è più evidente e andrà a favore dei metodi static.

Infatti immaginiamo di essere all’interno per esempio del main, se volessimo richiamare il metodo setTime(t) e il suo alterego setTime2(t) dovremo fare due cose differenti

|  |  |  |
| --- | --- | --- |
|  | setTime | setTime2 |
| Come si richiama | int t= 5;  Lib\_Funzioni lf = new Lib\_Funzioni();  lf.setTime(t); | int t= 5;  Lib\_Funzioni.setTime(t); |
| Risultato | In questo caso verrà creata un’istanza dell’oggetto quindi riservandogli una porzione di memoria ad esso e a tutte le variabili in esso contenute (potrebbero essere milioni e non utilizzate). | In questo caso non viene memorizzato niente e viene passato il parametro t direttamente. |

Ovviamente come avevamo detto in precedenza setTime2 non funziona per le limitazioni dei metodi statici però si può pensare a un metodo che non soffre di quei problemi come per esempio potrebbe essere:

**public** **static** **void** stampaASchermo(**float** variabile){

System.*out*.print(variabile);

}

Quindi sappiamo che i metodi statici ci limitano nelle operazioni ma vanno pesare molto meno sulla nostra ram per cui è conveniente utilizzarli dove possibile.